Winter 2012

Two-dimensional unsteady flow visualization by animating evenly-spaced streamlets

Zhengang Hong
University of New Hampshire, Durham

2D UNSTEADY FLOW VISUALIZATION BY ANIMATING EVENLY-SPACED STREAMLETS

by

Zhengang Hong

B.S., Chendu University of Technology, 2007

M.S., Chendu University of Technology, 2010

THESIS

Submitted to the University of New Hampshire

in Partial Fulfillment of

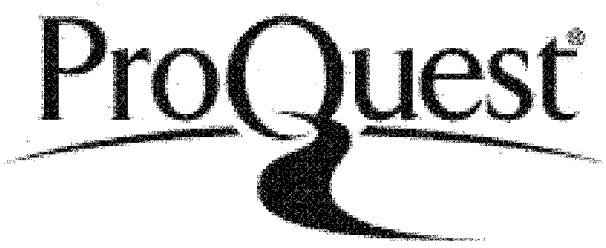
the Requirements for the Degree of

Master of Science

in

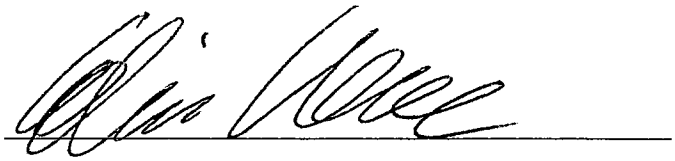
Computer Science

Dec, 2012



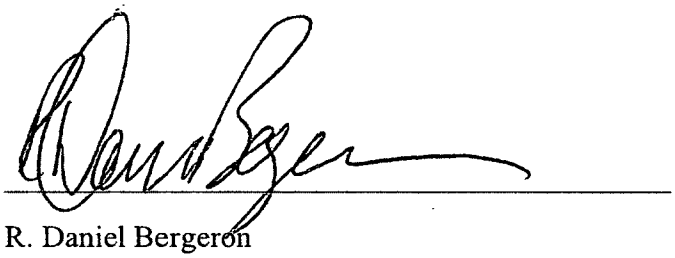
ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

This thesis has been examined and approved.



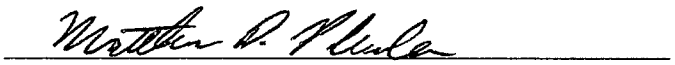
Thesis Director, Colin Ware

Professor of Computer Science



R. Daniel Bergeron

Professor of Computer Science



Matt Plumlee

Affiliate Asst. Professor of Computer Science

Date

DEDICATION

To my Fiancee, Jingwen, my Parents and my Brother.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Professor Colin Ware, for his thorough guidance, encouragement and support. I would like to thank other thesis committee members, Professors Dan Bergeron and Matthew Plumlee, for their expertise and guidance through this process.

I would like to thank all my dear friends at University of New Hampshire. They were always supporting me and encouraging me with their best wishes.

Finally, I would like to thank my fiancee, Jingwen Chai. She was always there cheering me up and stood by me through the good times and bad.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

2D UNSTEADY FLOW VISUALIZATION BY ANIMATING

EVENLY-SPACED STREAMLETS

by

Zhengang Hong

University of New Hampshire, Dec 2012

Flow visualization has been widely used to display and discover patterns and features in vector fields. Common applications include the representation of ocean currents and weather model data.

In this thesis, a flexible method for animating vector fields is developed, based on a generalization of a Poisson disc sampling method. The algorithm has two stages; in the first streamlets are drawn into an image buffer, larger than their intended size. Before they are drawn they are tested to see if they impact on already drawn areas; if they do, they are rejected. In the second stage the ones that pass the test are drawn normal size. The concept of a 3D streamlet object, which groups consecutive time step streamlets as a primitive rendering object, is introduced as part of a method for animating streamlets so that they have minimal overlap and show frame-to-frame coherence providing visual continuity when animating time

varying vector fields. Acceptance schemes that allow for occasional overlap between streamlets are explored and found to improve both the speed and the overall quality.

Both model data and real weather data are used to evaluate the method. The results show that the method produces good results and is flexible, allows for variable size and density of streamlets, and produces good results.

CHAPTER 1

INTRODUCTION

As society develops, science and technology create ever increasing volumes of data. People are surrounded by this sea of data and find it very difficult to interpret. As an important step of data analysis, visualization techniques have been introduced. The aim of visualization techniques is to use images to represent all kinds of abstract data so as to help people get a better understanding of these data. Scientific visualization, information visualization and visual analysis are three important branches in the area of visualization. Although scientific visualization has been used by scientist for centuries, modern scientific visualization relies on computer graphic technologies to visualize data that is generated in the process of scientific experiments.

1.1 Flow Visualization

Flow visualization is an important branch of scientific visualization. It has been widely used to display and discover patterns and features of vector fields. In a vector field, each position has been assigned a vector to represent the direction and magnitude attributes. Vector field data are usually generated from digital simulation models or experimental monitoring instruments. Flow visualization techniques are intended to show the structure of a flow field and its dynamic evolving pattern. They can help computational fluid professionals find potential problems existing in mathematical models or computation processes. Flow visualization has also been widely used in the

areas of atmospheric physics and weather forecasting. By visualizing atmospheric measurements and computational results, the motion pattern of air currents and the formation, development and evolution of eddy currents can be observed, so as to better predict when storms will come. Flow visualization techniques also have other applications like turbulence modeling and ocean current simulations [1, 2].

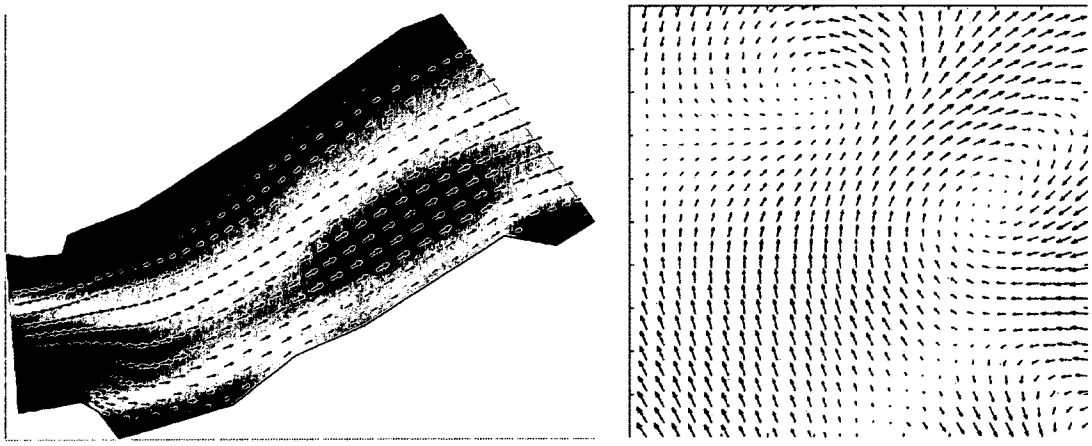
Vector fields can be categorized in terms of their spatial dimensionality into 2D, 2.5D and 3D. For 2.5D, it means flow visualization applied to surfaces in 3D [3]. With respect to temporal dimension, vector fields can either be unchanging over time, called *steady*, or time varying, called *unsteady*. Table 1 summarizes the categories proposed by Laramee et al. categories [4].

Spatial dimension / Temporal dimension	2 Dimensions (X & Y)	2.5 Dimensions (Surface)	3 Dimensions (X, Y & Z)
Steady	2D steady flow Visualization	Steady flow Visualization on surfaces	3D steady flow visualization
Unsteady	2D unsteady flow Visualization	Unsteady flow Visualization on surfaces	3D unsteady flow visualization

Table 1: Vector field visualization category in terms of spatial and temporal dimensions

1.2 Methods for Visualizing Vector Fields

McLoughlin et al. [5] created a taxonomy of four categories of methods used to visualize vector fields: *direct flow* visualization, *dense texture-based* flow visualization, *featured-based* flow visualization, and *integration-based geometric* flow visualization. According to their classification, *direct flow* methods include using color maps to show velocity and using arrows to show direction. However, arrow-based methods show limited information and may cause clutter when rendering three-dimensional data. Figure 1-1 shows two examples of using arrows to represent direction and using color map to show the velocity.



(a) Velocity vector and color map [6] (b) Icons on a regular grid [7]

Figure 1-1: Examples of direct flow visualization

Dense texture based flow visualization techniques use textures as a basis, and then modify the texture based on the local properties of the velocity field [8]. Figure 1-2 shows the result of using line-integral convolution method which adopts a 20 pixel wide box-shaped convolution kernel. The process was performed using a random texture as a

starting point with the value of each pixel randomly selected in the range of (0, 1). Texture based methods can generate dense visualization results and can show lots of detail. But they may cause clutter when visualizing three-dimensional data. Also most texture methods are ambiguous with respect to direction (they only show orientation) and they do not show speed effectively. Studies have shown that texture based methods are not as effective as other techniques, such as it lacks magnitude (speed) and direction information [7].

Figure 1-2: Line-integral convolution [7]

Featured-based flow visualization methods focus on flow features that the user cares about most rather than the whole dataset [9]. For example, critical points at the center of vortices are often emphasized [9], as shown in figure 1-3.

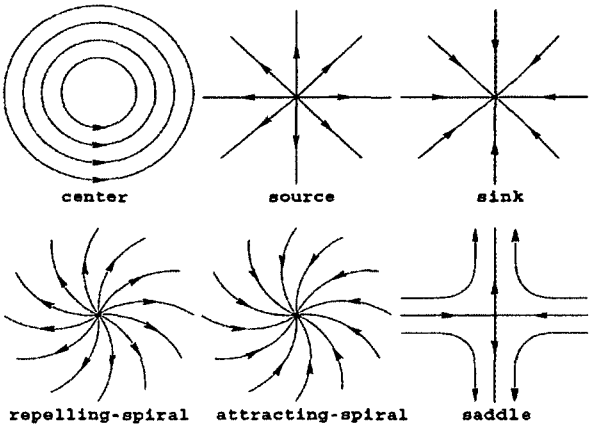


Figure 1-3: Different types of critical points possible in 2D flows [10]

McLoughlin et al. defined *geometric flow visualization* methods as using geometric objects to show the properties of vector fields. These geometric objects are created by integrating flow data. Streamlets, streamlines, streaklines, and pathlines are common geometric objects used for showing flow data. Figure 1-4 presents an example of using arrows, which are placed along the streamlines, to represent the wind vector fields over Australia.

Figure 1-4: Arrows showing the wind direction and magnitude over Australia. The arrows are placed along streamlines generated using the image-guided placement technique of Turk and Banks [5]

Figure 1-5 presents the comparison between the visualization results of using direct, texture-based, and geometric visualization methods. This helps us to get a better understanding of the pros and cons of each method as we discussed above. In the leftmost picture, arrow heads are placed on the surface of a ring model. Although this can gives viewers an intuitive impression about the flow field, it lacks details since the arrow heads are sparsely distributed. Meanwhile, continuity is lost between arrow heads.

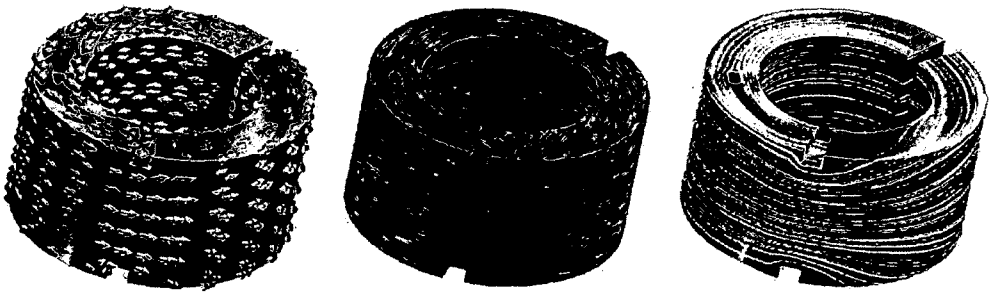


Figure 1-5: An example of circular flow at the surface of a ring to help illustrate flow visualization classification. [3]

The middle picture illustrates the result of using a textured based visualization method. It gives a complete and dense representation of the vector field, but lacks direction information. Speed is shown using color, but only on the top surface. The right most picture shows the result of using integral object streamlines to represent the attibute of the field. It clearly shows the continuity, but lacks of direction information. Again, speed is only shown clearly on the top surface.

In designing an effective flow visualization method, compromises between coverage and visibility need to be considered. The ultimate goal is to produce a

visualization that is perceptually efficient in the sense that flow direction, flow speed, and flow features such as eddies can all be clearly seen. Considerable work has been done on designing effective visualizations [11, 12, 13, 14, 15] for steady flow and some of this will be reviewed in later sections of this introduction. One of the conclusions of this research is that geometric methods using streamlines are often the most effective.

For two-dimensional *unsteady* vector field visualization, flexible methods are still required, and how to effectively visualize 2D unsteady flow using geometric objects is still an open issue [5]. We can extract more information from time varying vector fields and they have many applications ranging from meteorology to oceanography to automobile design. As we shall see, prior research suggests that equally spaced streamlines are perceptually the most efficient way of representing steady flow patterns [1, 2]. The goal of this thesis is to investigate stochastic methods for creating equally spaced streamlets in time varying flows, as a flexible method to allow for variable size and density of streamlets.

1.3 The Perceptual Evidence

Researchers in cognitive neuroscience have developed a model of contour perception [16] that may be applied to the problem of flow visualization [1, 2]. Contours oriented tangentially to flow direction are likely to be the best way of revealing flow orientation and supporting the task of understanding a flow pattern. This is related to the way the brain extracts extended contours from the visual environment. Contour lines have continuity which is an important factor for pattern recognition.

The rightmost picture in figure 1-5 shows the result of using contour lines to represent flow. Even though it can show the orientation, it fails to show the direction and speed attributes. Usually, a vector is decomposed into a direction and a magnitude. Ware [1] points out that the direction part can be further decomposed into an orientation and a vector sign for the aim of better understanding flow visualization. To represent a vector, the most common symbol is an arrow head, but it may cause clutter especially when arrows are close to each other. Variations like changing luminance along the line, or using head to tail shape can help to improve the visualization results by showing direction information. Other options suggested by Ware [1] have strong asymmetry features as shown in figure 1-6. Placing these objects "head to tail" along the contours of a vector field can help the user discover flow patterns effectively, because it combines the merits of showing orientation, direction, magnitude, and contours. Another important factor in showing flow effectively is the density of contour lines. High density distribution easily causes clutter and sparse distribution may loose important details. Flexible density is an important feature when designing algorithms to show complex flow patterns.

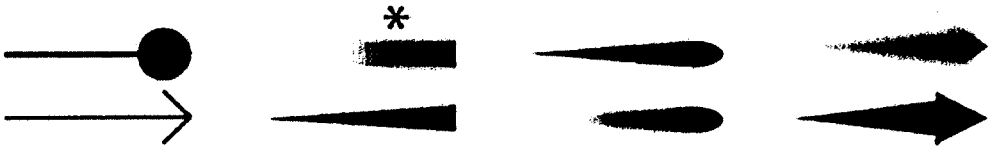


Figure 1-6: Various shapes to show direction

1.4 Streamlines and Streamlets

The focus of this thesis is on the use of streamlines which belong to the McLoughlin et al.s geometric flow visualization category. Streamlines are lines tangential to a vector field at every point along their length. The easiest way to integrate the streamline based on the vector field is to adopt the Euler method. For a 2D vector field, suppose at point (X_n, Y_n), the vector value is (U_n, V_n). The next point in the flow is determined by the vector value of the previous point. So,

$$X_{n+1} = X_n + U_n \cdot \Delta_t$$

$$Y_{n+1} = Y_n + V_n \cdot \Delta_t$$

This formula looses accuracy when dealing with curves. In order to improve the accuracy, we can decrease the step size or adopt a second (or higher) order Runge-Kutta method. The second order Runge-Kutta method calculates a temporaty position (X_{temp}, Y_{temp}) based on the Euler method and then gets the vector value (U_{temp}, V_{temp}) at that point. It then uses the average vector value between point n and the temp point $((U_n + U_{temp})/2, (V_n + V_{temp})/2)$.

$$X_{n+1} = X_n + ((U_n + U_{temp})/2) \cdot \Delta_t$$

$$Y_{n+1} = Y_n + ((V_n + V_{temp})/2) \cdot \Delta_t$$

Both the Euler method and the Runge-Kutta method adopt a fixed step size integrators. Adaptive step size integration methods can be even more accurate.

Streamlets are short streamlines, often used in animation. Using streamlets is an easy and effective way to capture the pattern of the vector field. Streamlets produce better visualization results than long streamlines when we animate time varying vector fields because they can have better frame to frame coherence [1, 2].

Streamline placement through seeding strategies can critically influence the appearance of flow data. Turk and Banks [11] proposed an image-guided streamline placement method to generate a specified density of evenly spaced streamlines by calculating an energy function in an iterative procedure. Jobard and Lefer [12] introduced an evenly spaced streamline placement algorithm which is much faster than the Turk and Banks method and can generate flexible results ranging from texture-like to hand-drawing styles. In their algorithm, they set the minimum distance (d_{sep}) requirement between nearby streamlines to control the density. The control is performed in the process of generating new streamlines. During the generating process, a new sample point is valid if the distance between the position and any existing streamline is no less than d_{sep}. If this new sample point fails the test, then the streamline integration stops in this direction (the method integrates the vector field from a sample position forward and backward simultaneously). In terms of speeding up validating of new sample positions, their algorithm requires that the sample points on the streamline be evenly spaced, and the distance should be smaller than d_{sep}. They divide the vector field into Cartesian sub-grids and the width and height of the cell is d_{sep}. So when a new sample point comes out, it only needs to be compared with the points in the same cell or neighboring cells. They

also provide another distance d_{test} used to define the minimum distance between the seed points and streamlines. d_{test} is a percentage of d_{sep}. Their results show that when d_{test} is half of d_{sep}, long streamlines are produced that provide good visualization results. Pseudo code for their algorithm is listed in table 2. Figure 1-7 gives an example of generating new streamlines based on the starting streamline.

```
Compute a streamline with sample points and put it into queue
while (queue is not empty, get a streamline from the queue)
    for (each sample points in current streamline)
        Generate a candidate point which is d_sep away from current sample point
        if (the candidate point is a valid control point, at least d_sep. from all streamlines)
            Compute a streamline starts with control point and put it into queue
    End for
End while
```

Table 2: Pseudo code of Jobard and Lefer's evenly spaced streamline placement algorithm

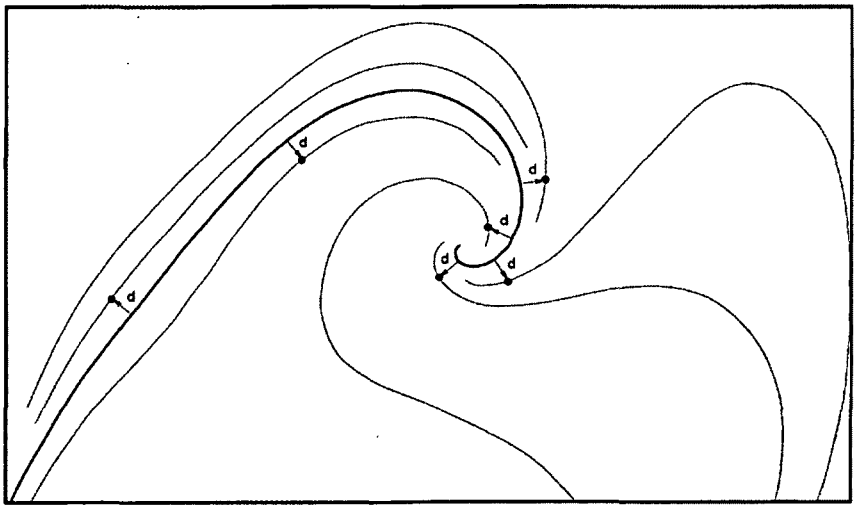


Figure 1-7: Generating new streamlines. Streamlines are derived from the first (thick) one by choosing seed points (circles) at a distance d=dsep from it. [12]

Verma et al. [10] discribe a streamline method to emphasize flow patterns around critical points. They preprocess the vector field to identify types of critical points and seed around each point with a template pattern of seeds that is different for each type, e.g. expanding vortex, saddle and contracting vortex. To fill the sparse areas, they adopt a random seeding algorithm based on Poisson disk distribution. Their method generates evenly distributed streamlines and does not miss important critical points.

1.5 Streamlines in Unsteady Fields

Steady vector field visualizations show the flow pattern at a certain time and are unable to represent the evolution or dynamic features of time dependent data. The problem with generating streamlines in time varying flow is having sufficient frame-to-frame coherence. Simply using Jobard and Lefer's method [12] on successive frames produces sets of streamlines that do not correspond between frames and the result is a flickering image. To produce a smooth animation it is necessary that streamlets have smooth transitions from frame to frame as far as possible.

Jobard and Lefer [17] present a method to visualize unsteady flow by animating evenly-spaced streamlines. They adopt a feed forward algorithm to correlate consecutive time steps. The first step is to generate a set of equally spaced streamlines based on their original algorithm. Following this they compute a set of streamlines for the next time step using the same seed points. For each streamline in time step n, a best matched corresponding streamline in time n+1 is selected from a group of candidate streamlines (for each control point in a streamline at time n, we can generate a streamline at time n+1

seeded from that control point). The criteria they used to select the best matched streamline is based on the average distance between all pairs of corresponding sample points along streamlines. Poor matches are discarded and the gaps are filled in using branches off the original streamlines. This method can still result in poor correspondences as illustrated in Figure 1-8 [17]. The streamlines are distributed with high density. From the picture we can easily tell that there is a lack of frame-to-frame coherence in some parts of the image. In addition the method cannot be easily generalized to deal with variable spacing of elements.

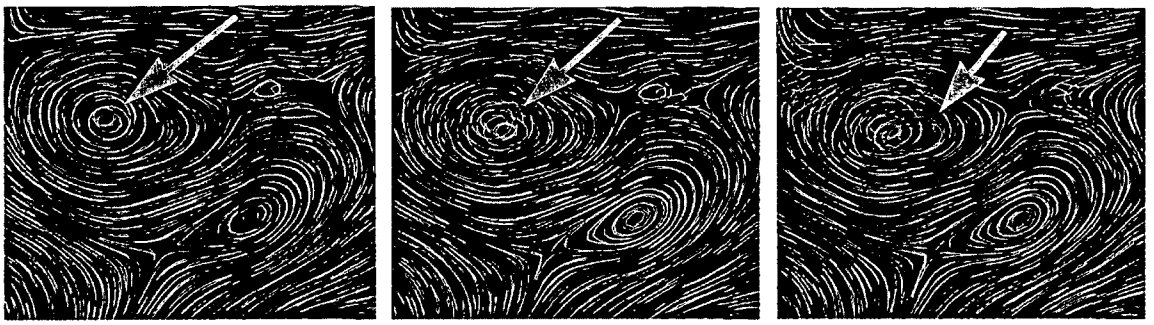


Figure 1-8: Three consecutive frames generated using Jobard and Lever method [17].

The solution proposed in this thesis is based on a generalization of methods designed to produce Poisson disk distributions. We applied and improved the Poisson disk distribution stochastic method into our research in order to generate evenly distributed streamlets with any density level. By using this very simple and easy method, we get very good results. Also, the method is flexible, making it possible to use various streamlet shapes as well as varied streamlet spacing as a function of properties of the vector field, such as speed. For example, streamlets can be more densely spaced in

regions of high speed flow. Additional refinements include streamlets fading in and fading out to improve the frame coherence, and parallel seeding to improve the speed.

In chapter 2, we introduce the Poisson disc stochastic method and show how it can be generalized for the purpose of vector field visualization so as to generate evenly distributed streamlets. Examples are given to show that the generalized Poisson disk distribution method is very flexible and can support various geometric shapes and even time varying shapes by adjusting the variables like distance between streamlets, streamlet object's shape, streamlet duration time, etc. In Chapter 3 the 2D streamlet method is generalized to support time varying streamlets. As a method for generating smooth animation, the basic algorithm is developed and methods for increasing its efficiency are explained. Experimental results and analysis for applying the unsteady flow visualization algorithm both to model data and real weather data is presented. Chapter 4 gives conclusions and possibilities for future research.

CHAPTER 2

GENERATING EVENLY SPACED STREAMLETS

Stochastic sampling methods have many applications in computer graphics including solving aliasing artifacts in ray tracing and ray casting algorithms [18]. In this chapter we first introduce a particular stochastic sampling method, the Poisson disk distribution. We introduce a novel method for producing Poisson disk distributions, then show how it can be extended to become an efficient and flexible method for generating evenly distributed streamlets for flow visualization.

2.1 Poisson Disk Distribution Stochastic Method

Aliasing artifacts occur in computer graphics when the regular grid of pixels creates unintended visual effects, such as "jaggies" at edges, or moiré patterns when a fine mesh is rendered. Sampling methods were introduced in this area to solve aliasing problems. Super sampling and adaptive sampling are two approaches to reduce the aliasing effect [18]. Super sampling improves the sampling rate (like using multiple samples in one pixel). Adaptive sampling pays more attention to areas where needed, such as places of rapid changes [19]. Both of these two methods alleviate the aliasing effect rather than eliminating the problem.

Stochastic sampling was introduced by Cook to solve the aliasing problem [18]. Unlike super sampling and adaptive sampling methods which are performed on regularly

spaced locations, the stochastic sampling method belongs to the non-uniform sampling category and it replaces aliasing with featureless noise. This work was inspired from research done by Yellott [20], who found that a nonuniform distribution of sample location distribution in the human eye can help to avoid aliasing. This nonuniform distribution was called Poisson disk distribution. To be more specific, a Poisson disk distribution is a random 2D distribution with the restriction that the distance between any two samples is greater than a certain value [18]. Poisson disk distributions have many applications in computer graphics, including object distribution. By using the Poisson disk distribution method, we can make sure no two objects are closer than a minimum distance.

2.2 Dart Throwing Technique

The most basic method for generating a Poisson disk distribution is the dart throwing technique, proposed by Cook [18]. The method involves repeatedly and randomly generating sample locations (metaphorically throwing darts) in the sampling region. If a dart is too close to an existing dart (determined by iterating over the existing darts list), it is removed. Otherwise it is identified and its location is stored. Darts are thrown until the sampling region is full. The pseudo-code for his algorithm is shown in table 3.

```
Npoints = 0;
while (empty space)
{
        p = rand(x,y);
        newpoint = true;
        For(i=0 to Npoints)
                If (dist(p, points[i]) < R)  newpoint = false;
        if (newpoint)
        {
                points[Npoints] = p;
                ++ Npoints;
        }
}
```

Table 3: Pseudo-code for dart throwing technique

This algorithm is inefficient because every new candidate point must be compared to all previous points in the set. Also, as the space becomes full it can take many new tests to find an empty spot.

Another method for creating non overlapping distributions of non-circular shapes involves first creating a set of random points on a plane (like a Poisson distribution) [21, 22]. Objects are assigned to these points and through an iterative process. The objects are moved according to forces generated through proximity with neighboring objects. Some points may be deleted or added as needed. Because of the iterative process, this method can also be slow.

2.3 Image Rendering Method

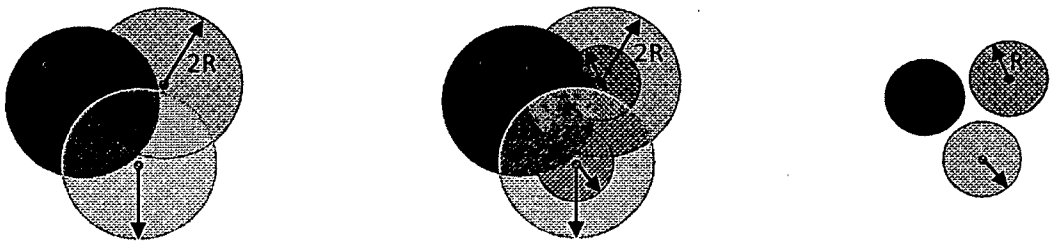
In this thesis, we explore an image rendering method that is a variation on the dart throwing algorithm. Our method has two stages. In the first stage, we start with a white

canvas (frame buffer); we repeatedly generate a random point on the canvas and read the color of pixel at that point to see whether this position has been colored or not; if not, we draw a circle on the canvas filled with a color other than white and add this point into our list. The filled circles have the radius 2R where R is the radius of intented rendering circles in our Poisson disk distribution. We keep generating new locations and drawing circles until the canvas is full (by calculating the canvas coverage rate) or after a certain number of attempts have failed. For the second stage, we draw circles with radius R centered on points in our list. This guarantees non intersecting circles on the canvas. The circle drawing method eliminates the inner loop of comparisons with all existing points since it only tests one pixel, reducing algorithm complexity by a factor of n.

Our image rendering algorithm can be easily implemented using OpenGL, a widely used application programming interface in computer graphics. In the first stage, we start with a white canvas, and this can be implemented as the frame buffer used for drawing in OpenGL. We initialize this frame buffer with a specific color as background color (white in our implementation). In stage one, once we get a sample point, we need to verify whether it is valid or not. We can use a function glReadPixels(), provided by OpenGL, to get the color at the location of sample point. Since we are drawing circles and filling them with a color other than the background color, the sample point is a valid point if the color we read is the same as the background color.

Figure 2-1 shows the steps to generate non-intersecting circles. We start by generating circles with 2R. For instance, we find a random uncolored point on a canvas

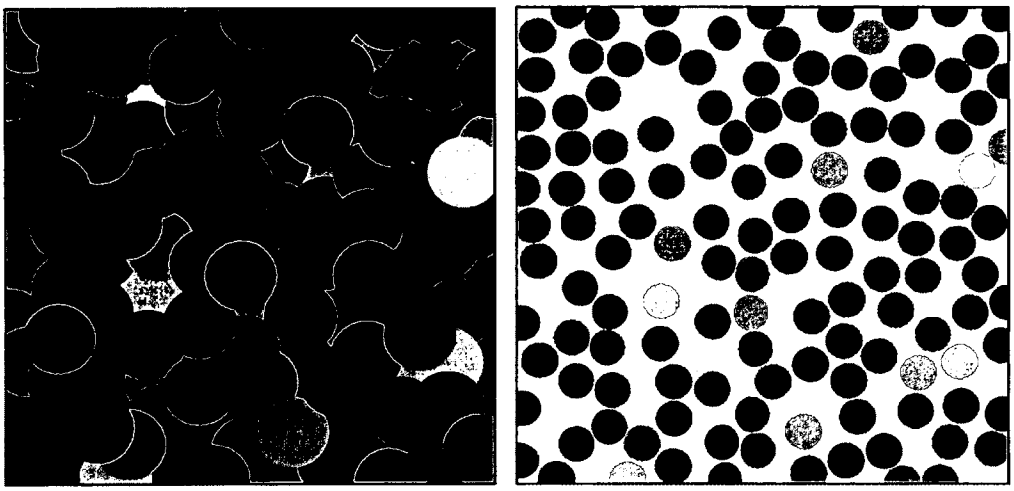
and draw a circle centered with that point. The radius of the circle is 2R then we fill the circle with red color. Then we randomly find another uncolored space and draw a blue circle with 2R radius. In figure 2-1(a), the center of this blue circle is drawn as close as possible to the red circle, so that we can show that the distance between the centers is at least 2R. The same procedure is used to generate the green circle. In stage two of our algorithm, we draw smaller circles centered with the points stored but with radius less than or equal to R. Since we know that the distance between any two points is at least 2R, this ensures that the smaller circles will never intersect with each other. The result is shown in figure 2-1(c).



(a) Circles with 2R radius (b) Concentric circles (c) Circles with R radius

Figure 2-1: Steps to generate non-intersecting circles

The result of using our method to generate circles having a Poisson disk distribution is shown in figure 2-2. From this figure, we can see that the circles are almost evenly distributed and no two circles intersect.



(a) Image rendering 2D (b) Evenly distributed intersection free circles

Figure 2-2: Using image rendering method to generate non- intersecting evenly distributed circles

The pseudo-code for our method is shown in table 4.

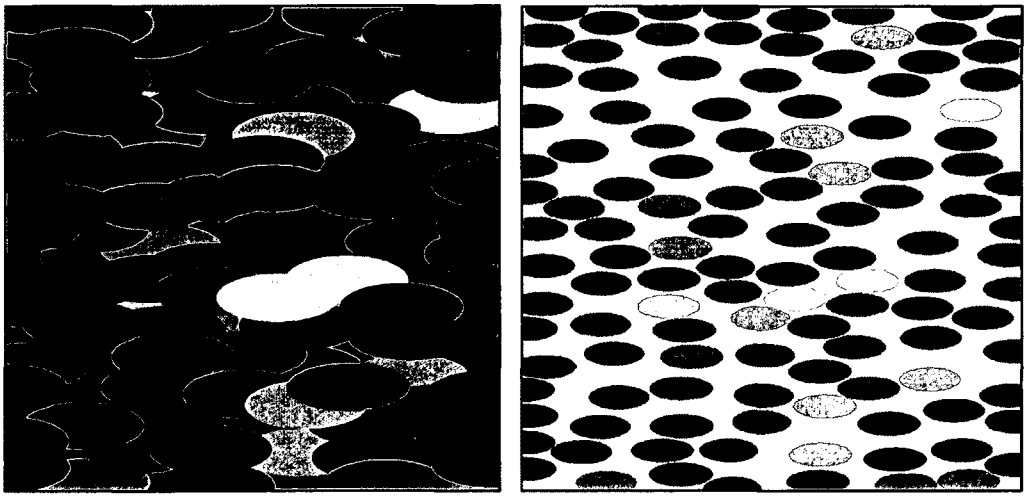
```
STAGE 1:
Npoints = 0;
Clear image buffer to white;
while (empty space)
{
        p = rand(x,y);
        if (buffer(p) is not colored)
        {
                points[Npoints] = p;
                draw disk of radius 2R at point p.
                ++Npoints;
        }
}

Clear image buffer to white

STAGE 2:
for (i = 0; i < Npoints; i++)
        Draw disk of radius R at points[i];
```

Table 4: Pseudo-code for image rendering method

The image rendering method can be easily generalized to deal with ellipses as shown in figure 2-3. Figure 2-3(a) shows the result of generating ellipses with major axis 5R and minor axis 2R to cover the whole area. Figure 2-3(b) shows the result of drawing ellipses with major axis 2.5R and minor axis R size based on the positions in figure 2-3(a).



(a) Image rendering 2D (b) Evenly distributed intersection free ellipses

Figure 2-3: Using image rendering method to generate non- intersecting evenly distributed ellipses

Notice the fact that when the canvas is close to being filled, the algorithm slows down because finding an uncolored spot takes much more time. At this point, instead of placing sample points randomly, a final pass that systematically looks for holes and fills them in could be implemented. For example, we can read in the frame buffer and sample every Kth pixels to find an uncolored spot, K >= 1. This step could improve the speed of our algorithm dramatically.

So far, we have applied our algorithm to distribute primitive shapes like circles and ellipses sharing the same orientation. But what if the orientation of candidate ellipses is different from the original one? In this case there can be overlap using the basic method. Figure 2-4 gives an example. The solid red ellipse has been validated, the black outline ellipse centered with the black dot is the newly generated one. Even though the black dot passes the test, overlap exists. In general, the problem will be greater when there is a large mismatch in orientation between nearby ellipses and this means that with smoothly varying flow patterns there will be relatively few collisions.

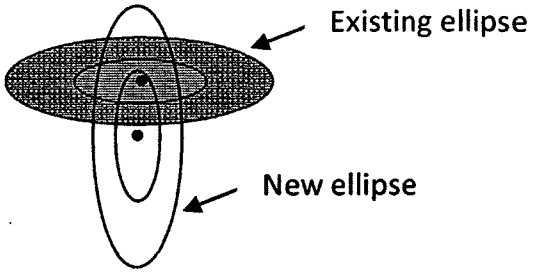


Figure 2-4: Adjacent ellipses with different orientation

There are a number of methods for solving the problem. The first is to allow a certain amount of contact. The second is to simply use circles that completely enclose the symbol, whatever their shape in creating the distribution, but this will place ellipses further apart than they need to be.

A third solution is to enlarge the ratio of ellipses between two stages. From the central point to the edge of the shape, we can find a longest path d_{max} and a shortest path d_{min}. If the ratio of the size between the drawing shape and the actual shape is larger than

the ratio of $(1 + d_{max} / d_{min})$, then intersection could be avoided in the final result. This method works not only with ellipses but with any convex shape.

Figure 2-5 shows the result of eliminating the intersection effect by enlarging the ratio of shape size in stage one and stage two. In figure 2-5(a) the ratio of edges between the larger square and smaller square is 2 and it causes intersection when the black box and the blue box is in perpendicular orientation (worst situation). Since the ratio of d_{max} / d_{min} for square is $\sqrt{2}$, we enlarge the ratio of edges between two stages to $(1 + \sqrt{2})$, the result shows in picture 2-5(b), intersection problem got solved.

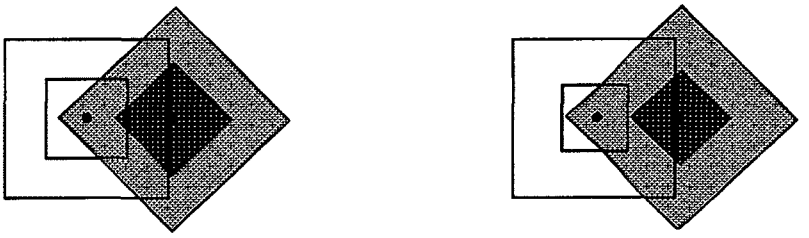


Figure 2-5: Enlarge the ratio of size of shapes in stage one and stage two could help to avoid intersection

The fourth solution involves adding test points. Rather than just checking the central point, we add more test points along the major axis of an ellipse. As shown in Figure 2-6, we have three test points (red points) that need to be validated. This can make sure that the ellipses in stage two will not intersect with each other and will ensure more closely spaced glyphs.

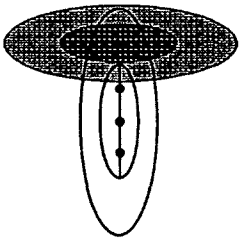


Figure 2-6: Multiple test points need to be validated

By setting the maximum distance between any two consecutive test points to be the width of the minor axis of rendering ellipse of stage two, even very long narrow ellipses can be dealt with (as shown in figure 2-7).

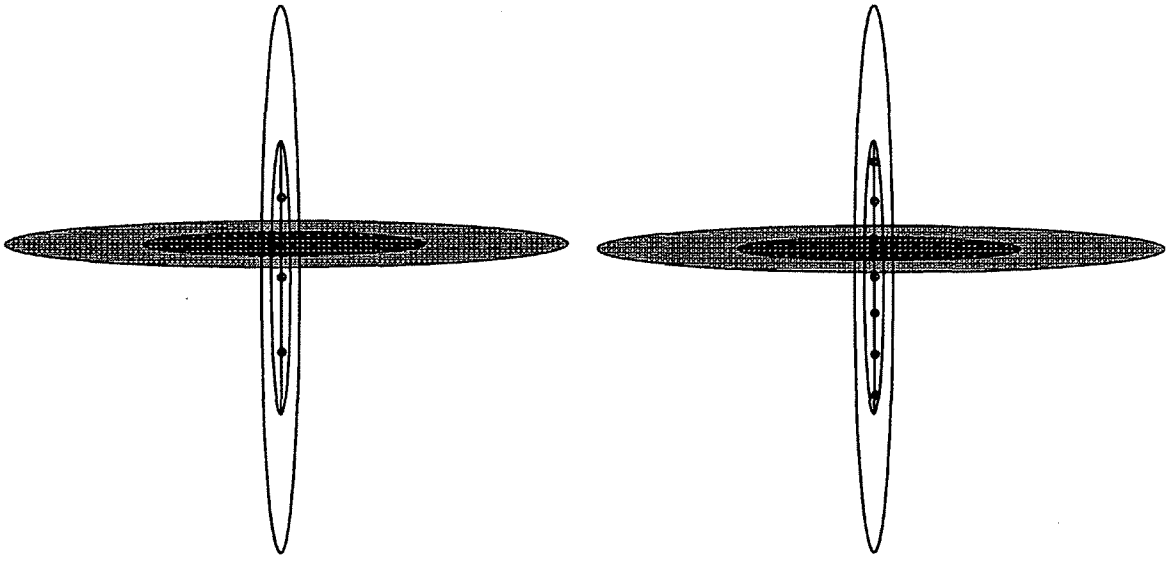


Figure 2-7: Add more test points to avoid intersection

2.4 Generalized Poisson Disk Method for Arbitrary Shape Objects

The Poisson disk method can easily be generalized further to non-convex shapes. In chapter one, we mentioned that streamlets are widely used in visualizing vector fields.

A streamlet can be created by integrating a vector field in a sequence of steps, creating a set of control points. Based on the control points, a streamlet can be constructed into a set of primitive shapes which work with our image rendering algorithm. Figure 2-8 illustrates the concept of decomposing streamlets into primitive shapes.

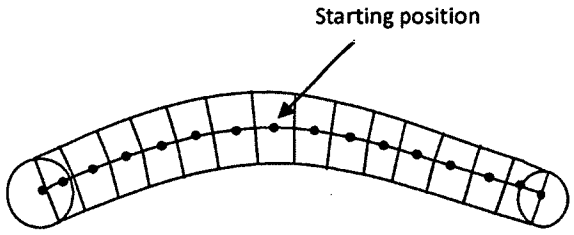


Figure 2-8: Formation of a streamlet

As figure 2-8 illustrates, the integration process starts from a random uncolored position. The control points can also be used as a test points in our non-intersecting algorithm. As the streamlet is generated, points move according to the vector field bi-directionally (both forward and backward) by N steps. For every new control point, the color at this position is tested. If any point fails the test, the integration process stops and this streamlet is abandoned. The streamlet is accepted if all the control points have not been previously colored. If a trace encounters the edge of the data field it halts and all remaining points are drawn at the edge location.

The method also allows for a relaxed acceptance criterion, allowing for some occasional degree of streamlet intersection. In this case not all of the test points need pass the coloring test. Relaxing the testing criteria can allow us to get denser coverage. The

acceptance rate (the ratio of valid test points and total test points) could be set as the criteria to accept a streamlet.

Once we find all the control points are valid candidates. We need to render the streamlet. Based on the location and vector value of the control points, we can calculate two points that are *W*/2 away from the control points (*W* is the width of streamlet). Then we use GL_TRIANGLE_STRIP in OpenGL to connect all the vertexes on both sides of the curve (composed by control points) separately. We also put circles on both end to avoid the situation that two adjacent streamlets append together. This could also provides a smooth ending, to get a better aesthetic effect. The streamlet shape is illustrated in the Figure 2-9.

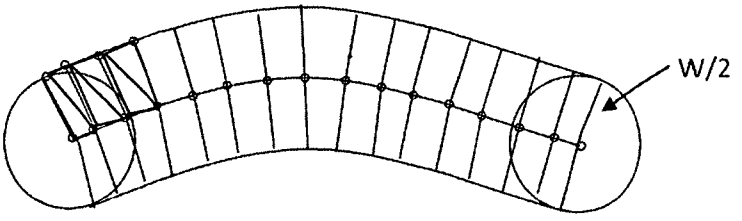


Figure 2-9: Drawing streamlet based on control points

As before the algorithm has two stages with different widths in each. *W* represents the width in first stage, and *w* represents the width in stage two. Width *W* is two times bigger than the width *w*. Figure 2-10 shows the two streamlets have same orientation and they are not crossing even in stage one, so this situation is acceptable. To ensure that there is no streamlet intersection, the internal test points must be spaced less than *W* apart and the end test points must be less than (*W*-*w*)/2 from the end of the stage one streamlet.

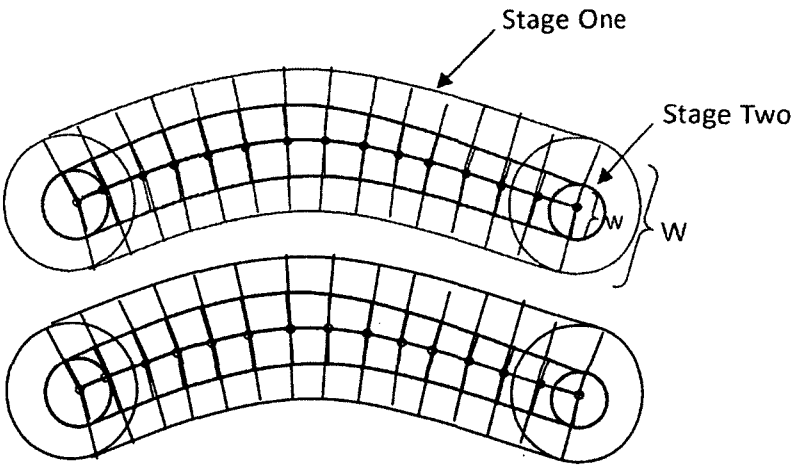
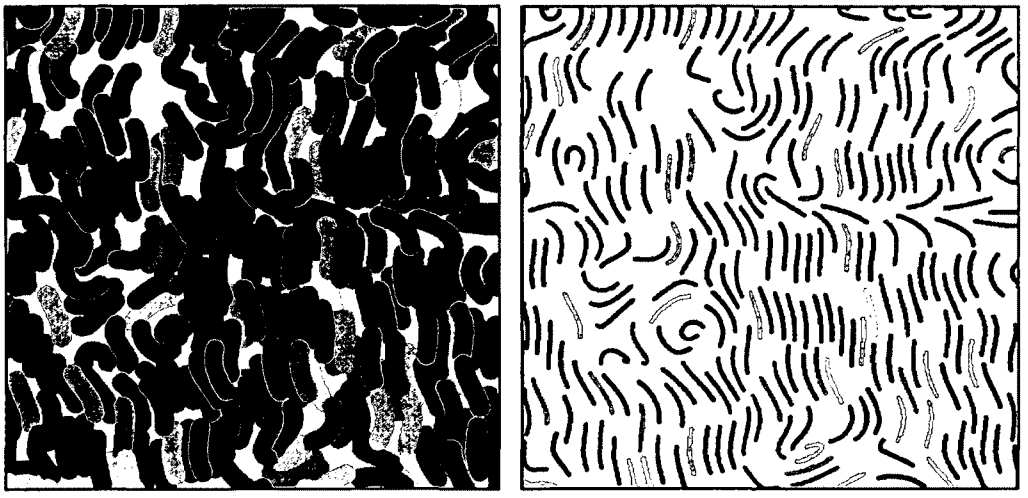


Figure 2-10: Two non-intersecting streamlets with same orientation

Figure 2-11(a) shows a pattern of streamlets generated from an artificially produced vector field. In stage one each has a width of 32 pixels and they are placed in a rendering area of 800 pixels by 800 pixels. Each streamlet has 20 control points with a step size of 4 pixels. We generate a random position first. If that position has not been colored yet, we treat that position as a starting point and then integrate from that position both forward and backward according to the vector value. In the process of integration, every control point in the streamlet is tested to see if it has been colored or not. If all the control points have not been colored, we then add the streamlet into the list, which is used in the final rendering (see Figure 2-11(b)). The ratio of widths between Stage one drawing and Stage two drawing is 4:1.



(a) (b)

Figure 2-11: Using Image rendering method to generate non-intersecting evenly distributed streamlets. (a)Image rendering 2D, (b) Evenly distributed intersection free streamlets.

In general, the minimum distance between any two streamlets can be calculated by the following formula:

Min distance = $W/2 - w$

where W is the width of the streamlet in Stage one and w is the width drawn in stage two.

From the above formula, we can adjust the value of W and w to easily determine the distance between streamlets so as to generate sparse or dense distributions.

In figure 2-11(b), we draw the same streamlets shown in figure 2-11(a), but with line width of 8 pixels in Stage two. So that we can make sure the distance between any streamlet is no less than $32/2 - 8 = 8$ (pixel).

Figure 2-12 shows the resulting streamlets distributions for different combinations of width W for stage one and width w for stage two of our algorithm. Figure 2-12(a) and 2-12(b) have the same streamlet width in stage one which is 16 pixels. But in stage two, they are given different widths (8 pixels for 2-12(a) and 2 pixels for 2-12(b)) to produce various effects. The rendering in 2-12(b) has the same number of lines as 2-12(a), but they are thinner, and this may be desirable if other information is to be shown between the streamlets. Figure 2-12(b), 2-12(c) and 2-12(d) have the same streamlet width but with different density. That is because in stage one, they have different streamlet widths which are 16 pixels, 8 pixels and 4 pixels. Respectively, these variations show that our algorithm is very flexible and easy to use to generate dense/sparse object distributions.

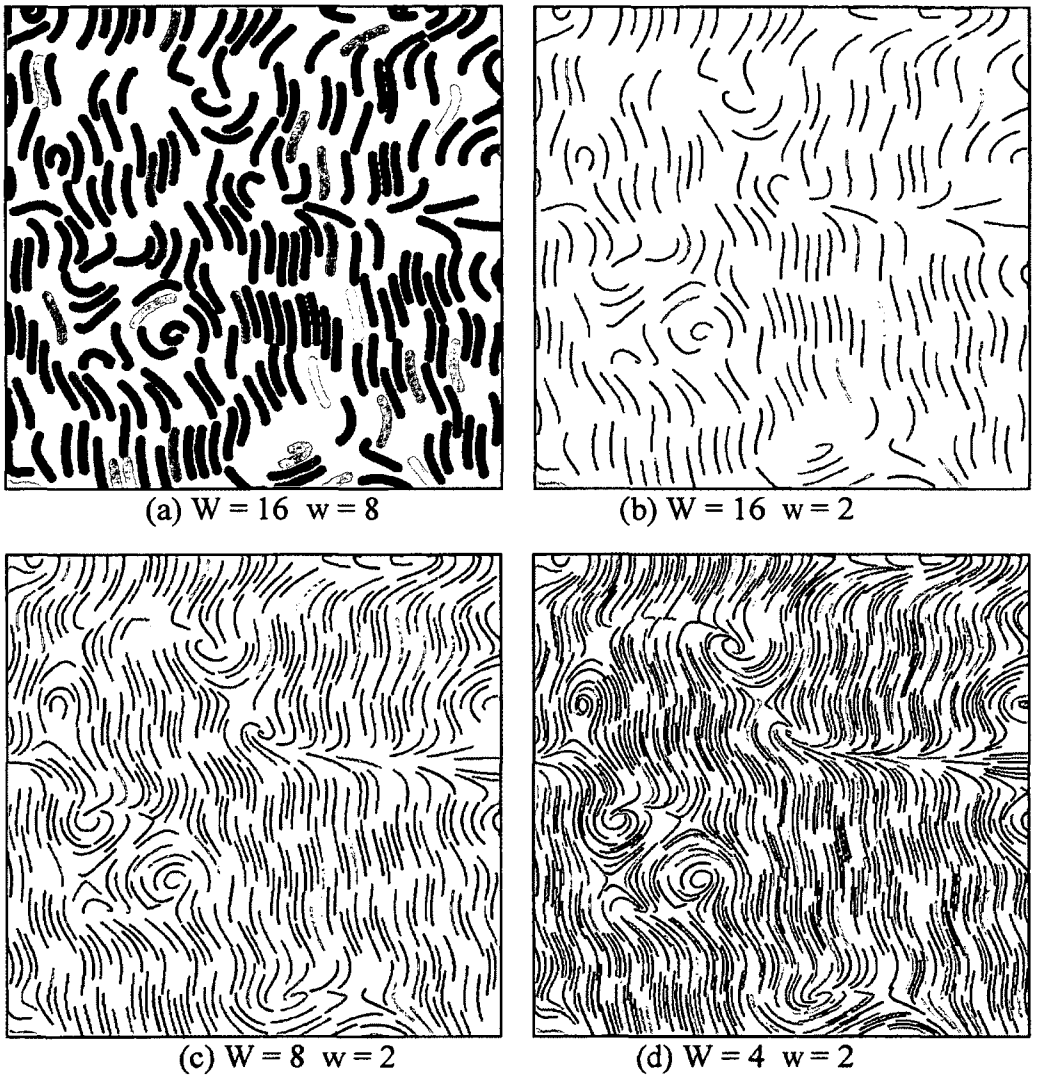


Figure 2-12: Various combinations of width W & w (pixel) to generate dense/sparse distribution effect

2.5 Variations of Streamlets

So far, we have used a basic streamlet symbol (fixed integration step size and streamlet width) as a primitive element to represent a vector field. But it only shows limited information. We can also use other attributes of a vector field such as magnitude and direction to determine streamlet shapes.

2.5.1 Width and Length

In our design, streamlets have the same number of control points, so the length of a streamlet is determined by the integration step size. We can either have a fixed step size or let it vary with the speed of vector field in the process of integration. Figure 2-13 shows how streamlet length can be made proportional to magnitude.

Figure 2-13: Streamlet length proportional to wind speed

Figure 2-14 illustrates the idea of letting the width of a streamlet increases with the higher speed. We keep the integration step size unchanged, but allow the width of the streamlet to vary with the magnitude.

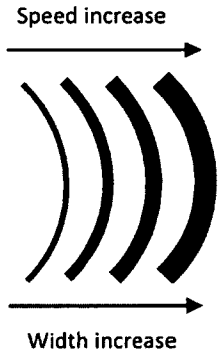


Figure 2-14: Streamlet width proportional to wind speed

Figure 2-15 combines the length and width factors together to provide an even better visual solution. Both the streamlet's length and width increases as the speed goes up.

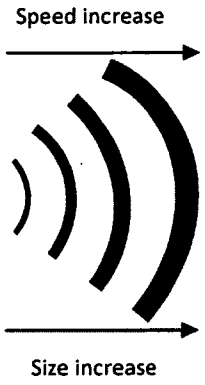
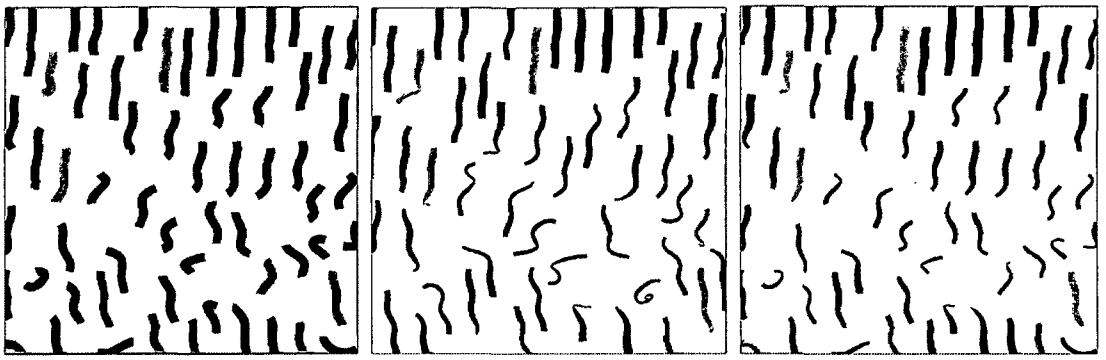


Figure 2-15: Streamlet size (both width and step size) proportional to wind speed

Figure 2-16 presents three visualization results of applying the above concepts to an artificially produced vector field. In figure 2-16(a), streamlets have the same width but with different length. This figure gives a general idea about which area has higher speed than other areas. But it misses the speed information for a single streamlet. Figure 2-16(b)

shows the result when the streamlet's width is proportional to speed. The changing of speed over streamlet can be observed, but the speed distribution for the entire area is not as straightforward as figure2-16(a). Figure 2-16(c) combines the width and length together for designing streamlets to get a better representation of speed.



(a) (b) (c)

Figure 2-16: Streamlet width and length various with vector field. (a) Length, (b) Width, (c) Both length and width

2.5.2 Streamlet Shape

As shown in section 1.3, asymmetric shapes give good visualization results in terms of showing the direction and orientation for a vector field. In this section, we implement three of those variations as shown in figure 2-17. Figure 2-17(a) lets the width of the streamlet keep increasing along its length. Figure 2-17(b) adds a circle head to 2-17(a) which can show smooth ending and give better aesthetic effects. Figure 2-17(c) adds an arrow head at the end.

(a) (b) (c)

Figure 2-17: Asymmetry streamlet shapes

In order to show both direction and speed information, we can map the magnitude to each of these shapes to get a better visualization effect. The results are shown in the following section.

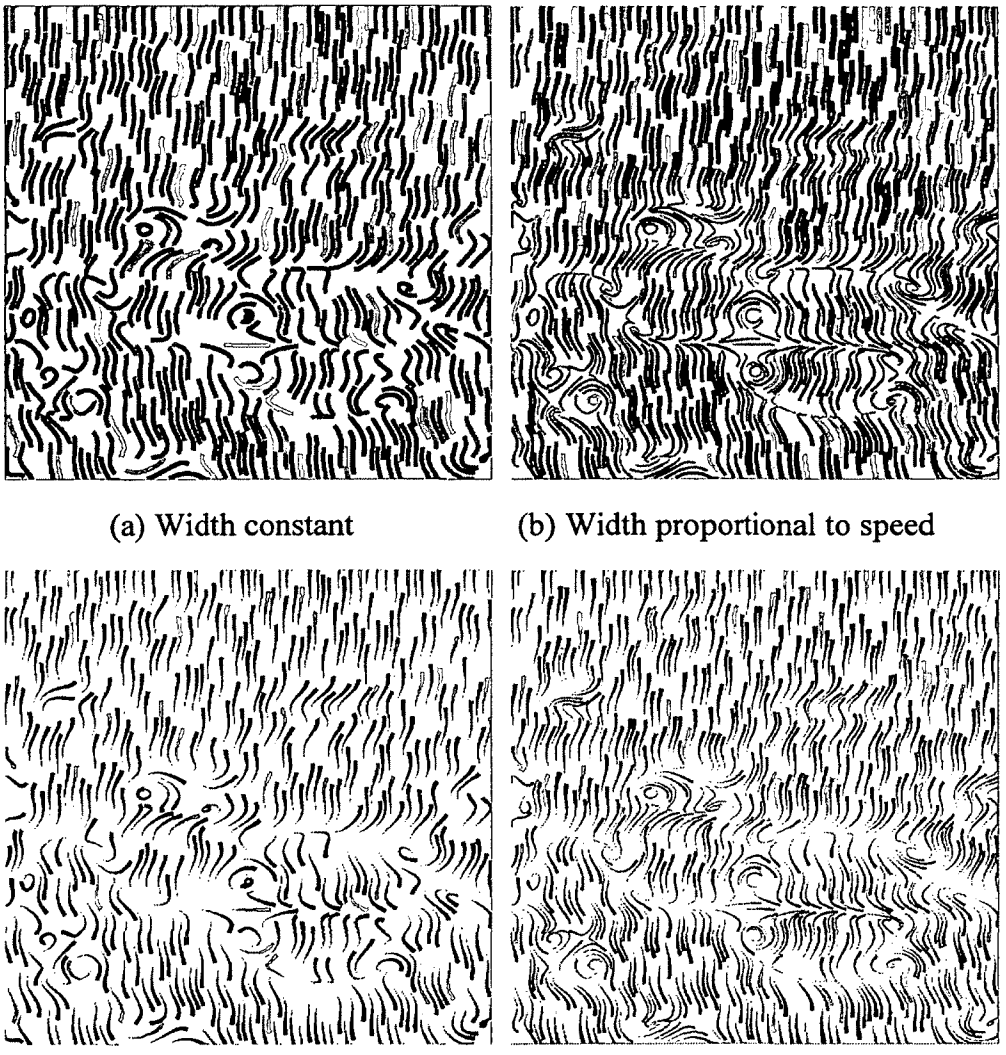
2.6 Results

As mentioned above, a streamlet can be designed based on attributes of the vector field. For example, the width can be proportional to magnitude, the length proportional to magnitude, or both width and length proportional to magnitude. Also the shape can be varied. For example, the width can increase along the streamlet to show direction, and a circle head or an arrow head can be applied. Considering these alternatives, we have 16 combinations in total. In this section, we show the result of applying our image rendering algorithm for all 16 cases.

2.6.1 Variable Width Fixed Integration Step Size

Figure 2-18 shows four variations all having the same streamlet length but with variable width. The size of the vector field is 800 pixels by 800 pixels. The generating parameters are as follows: Figure 2-18(a) W=16 pixels, w=8 pixels. Figure 2-18(b) W varies with speed (max = 16 pixels), w = W/2. Figure 2-18(c) W=16 pixels, w varies

along the length of the streamlet (1 to W/2). Figure 2-18(d) combines the properties of (b) and (c), the width is proportional to the speed and it also varies along the length of the streamlet. The acceptance rate is 90%. There are 20 control points in each streamlet, and the integration step size is 4 pixels.Notice that as the width decreases the spacing also decreases and the number of streamlets in a particular area increases.

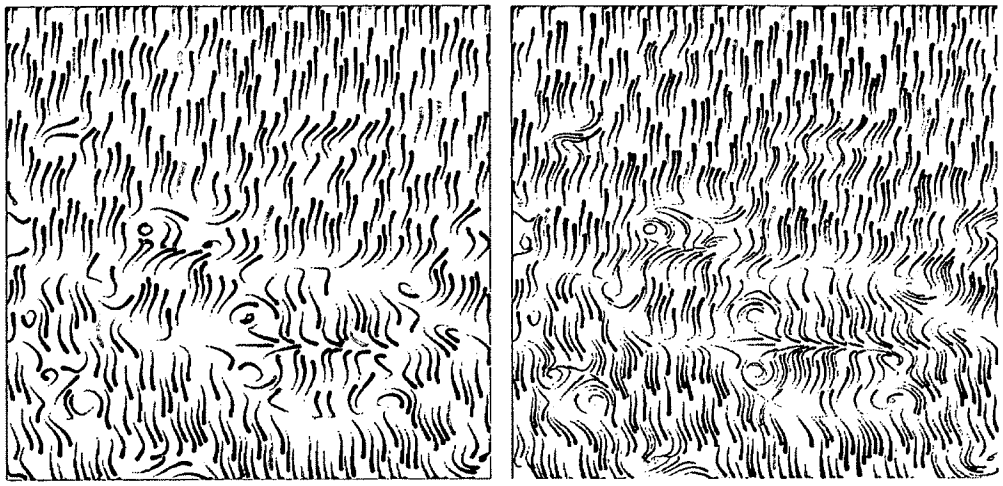


(a) Width constant (b) Width proportional to speed

(c) Width increases along direction (d) Width increases along direction and speed

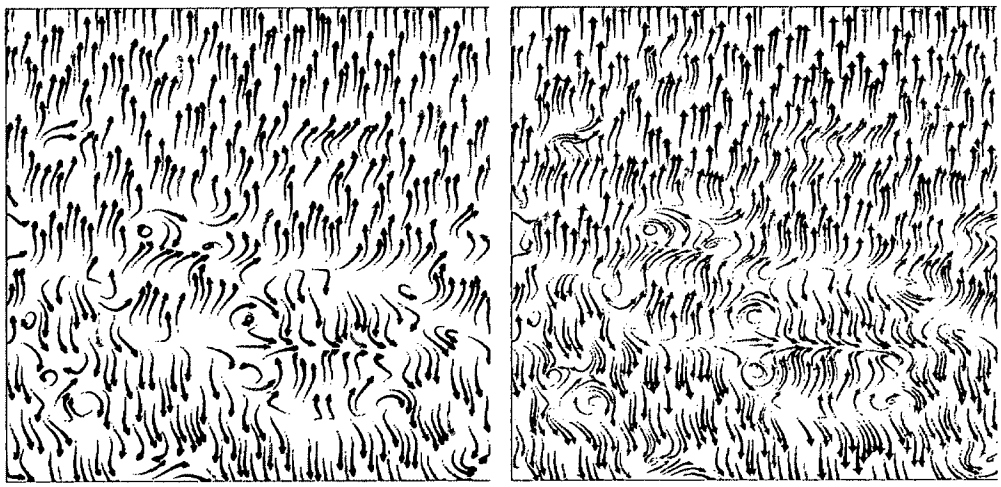
Figure 2-18: Width and spacing varies for fixed length streamlets

Figure 2-19 shows the results of modifying the streamlet shape by adding a circle or arrow heads. Figure 2-19(a) and Figure 2-19(c) have the same parameters as Figure 2-18(c). Figure 2-19(b) and figure 2-19(d) have the same parameters as Figure 2-18(d).



(a)Width increases along direction with circle head

(b) Width increases along direction and speed with circle head



(c)Width increases along direction with arrow head

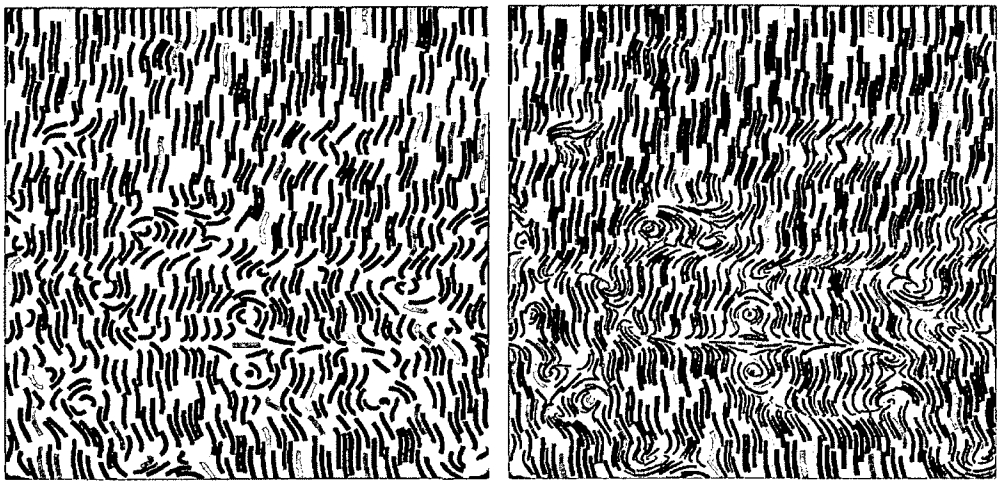
(d) Width increases along direction and speed with arrow head

Figure 2-19: Shape varies for fixed length streamlets

2.6.2 Length Proportional with Speed (Integration Step Size proportional with Speed)

Figure 2.20 shows a set of examples where the length of streamlets varies according to the speed (maximum integration step size is 4 pixeles). In order to accomplish this, we let the integration step size vary based on vector field magnitude. The rest of the parameters are the same as in section 2.6.1.

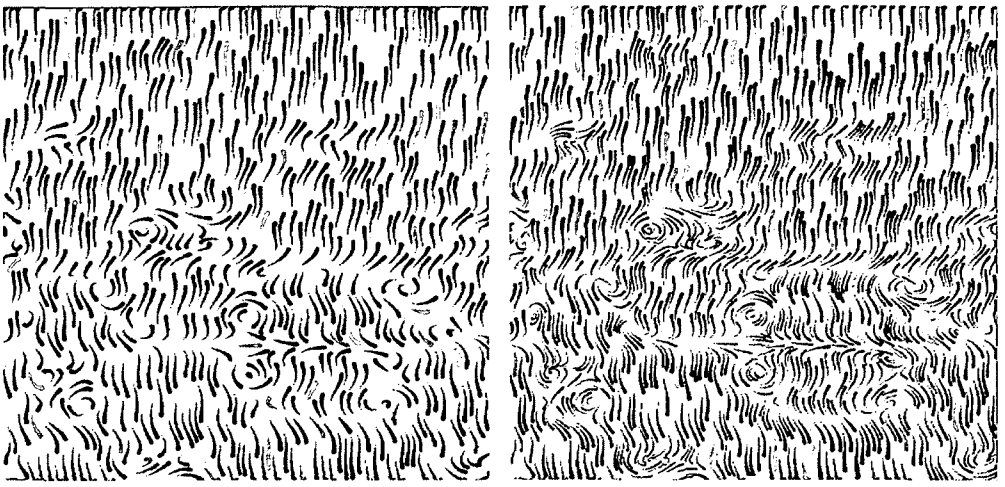
Figure 2-21 shows a set of examples with circle and arrow heads placed on the streamlets.



(a) Width constant (b) Width proportional to speed

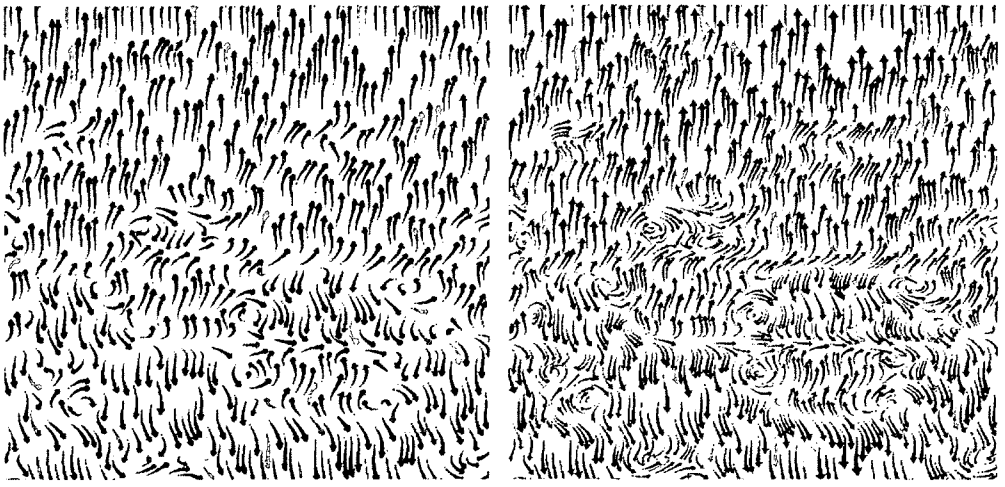
(c) Width increases along direction (d) Width increases along direction and speed

Figure 2-20: Width varies for varying length streamlets



(a)Width increases along direction with circle head

(b) Width increases along direction and speed with circle head



(c)Width increases along direction with arrow head

(d) Width increase along direction and speed with arrow head

Figure 2-21: Shape varies for varying length streamlets

CHAPTER 3

TIME VARYING STREAMLETS AND SMOOTH ANIMATION

In this chapter, we generalize our 2D streamlet distribution method to time varying vector fields in order to achieve a smooth animation effect. The goal is to enable viewers to observe how patterns are formed and diminished. The concept of a time varying streamlet object is introduced. Techniques including relaxed streamlet object acceptance criteria, sub-sampling and parallel seeding strategy, extrapolation, and streamlet object fading in and fading out are adopted to improve the seeding speed and to get better animation results.

3.1 Time Varying Streamlet Object

Animation is usually a good and straightforward method to show the evolution of a pattern. Figure 2-11 shows the 2D streamlet method applied to steady flow. To visualize unsteady vector flow and to show flowing patterns, a simple method would be to render each time step separately as before and then show them successively. Even though this method can show the changing of vector field over time, it has obvious drawbacks. Because we adopt the random seeding method to generate evenly distributed streamlets for each time step, there will be no correspondence between the placement of streamlets in successive frames. This will produce visual noise and flicking during animation. In order to have frame to frame coherence we propose the concept of a time varying streamlet object.

A time varying streamlet object involves grouping consecutive 2D streamlets together as a primitive 3D (space and time) object. These streamlet objects are seeded from the same position but integrated forward and backward separately at each time step. Fig. 3-1 illustrates the concept of a time varying streamlet object containing 5 time steps (the streamlet object is constructed by 5 consecutive streamlets.

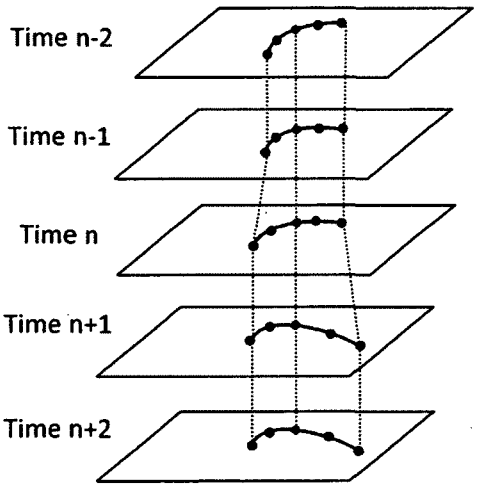


Figure 3-1: Time varying streamlet object with unchanged seeding position

In Figure 3-1, each streamlet is represented as a curve formed by connecting all control points.). In this example, from time step n, we trace 2 time steps forward (time n+1, time n+2) and 2 time steps backward (time n-1, time n-2). The green point in each time step is the seeding point (starting point), and this position remains unchanged in a sequence of time steps. For each time step, we integrate the 2D streamlet from the seeding point bidirectionally based on the vector field (blue points). Since the vector fields are consecutive outputs from a flow model, the result is visual continuity between successive streamlets. The overall effect of this procedure is to produce a set of

streamlets, each anchored about its center, that twist and turn as the vector field changes over time.

For time varying flow fields, there are two time related factors we need to consider when we seed a vector field. The first is that the vector field keeps changing over time so the flow pattern at different time steps is different. The other one is that a particle dropped in the flow at one time step will have moved forward on the next time step. If we want our streamlets to move with the flow, the seed points must move (advect) from one time step to the next. This is illustrated in Figure 3-2.

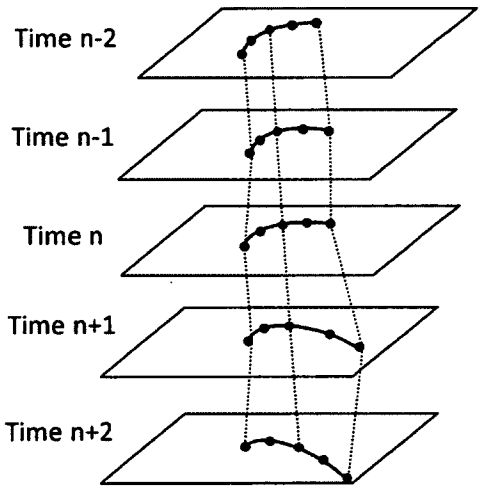


Figure 3-2: Time varying streamlet object with moving seeding position

3.2 Canvas Matrix Used for Rendering

In order to support the further generalization of the Poisson disc method, the two dimensional canvas described in the previous chapter is extended to become a space-time cube as illustrated in Figure 3-3. Instead of using a simple canvas, we create a canvas

layer for each time step. When we create a 3D streamlet object, we need to render a set of 2D streamlets on consecutive canvases simultaneously (each streamlet starts from the red point, and then integrates the vector field in both forward and backward directions).

As mentioned in the previous chapter, in terms of implementing the image rendering algorithm, we use the frame buffer provided by OpenGL as our rendering canvas. In order to support rendering multiple time steps, we divide the frame buffer into a canvas matrix where each cell represents the canvas for each time step. For a single step canvas, it can be defined in following structure:

```
struct Canvas
{
        int height;  // the height of vector field
        int width;  // the width of vector field
};
```

If we have N time steps which could be calculated by ROW times COLUMN. Then the canvas matrix could be represented as

```
Canvas canvasMatrix[ROW][COLUMN];
```

Figure 3-4 shows a canvas matrix for 16-time-step which is the frame buffer used for image rendering. This canvas matrix corresponds to the space-time cube shown in Figure 3-3.

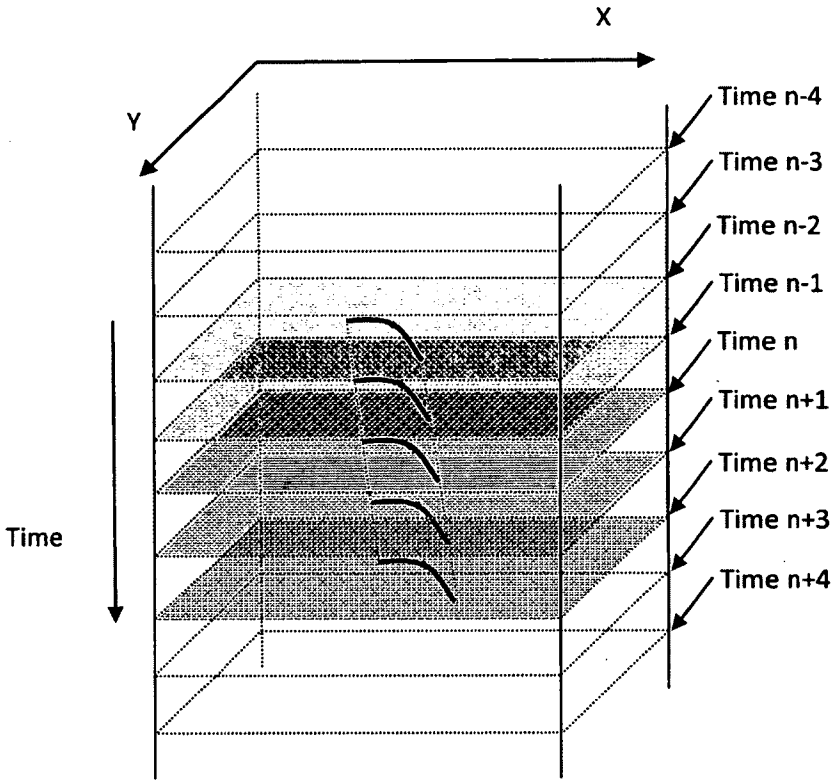


Figure 3-3: Model for space-time cube

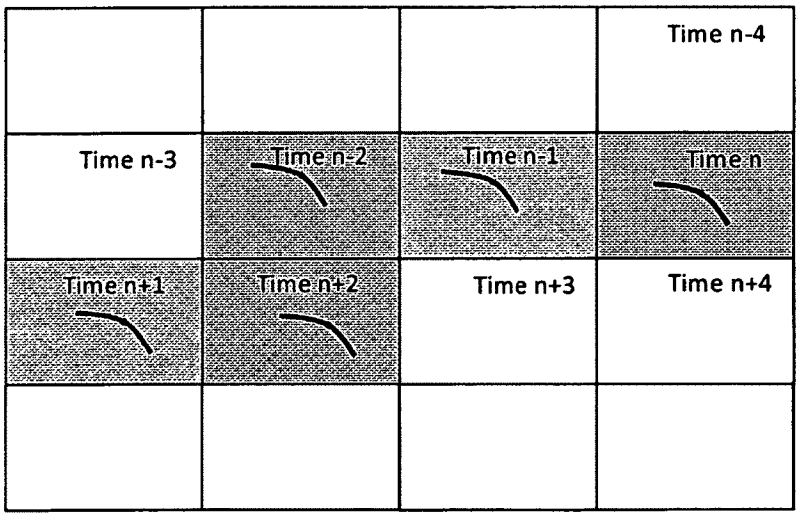


Figure 3-4: 16 time steps matrix used for image rendering

If we have N time steps (ROW by COLUMN) and the size of the vector field is WIDTH by HEIGHT, then we need to create a frame buffer whose width and height are ROW * HEIGHT, COLUMN * WIDTH. In the process of selecting seeds points, constructing streamlets, reading pixel colors and detecting collision, two sets of coordinates are adopted. To read a pixel's color or draw streamlets, it is necessary to convert x,y,t coordinates to canvas matrix coordinates. But when integrating a streamlet, local cell size coordinates are adopted. So a point's location must be transformed between those two sets of coordinates. For example, if we want to integrate a streamlet from position (x, y) in time t, $0 < x <$WIDTH, $0 < y <$ HEIGHT, $0 < t <$ N. The position where we draw this starting point in frame buffer should be (X, Y), where

$$X = (t \% COLUMN) * WIDTH + x$$

$$Y = (t / COLUMN) * HEIGHT + y$$

3.3 The Algorithm

In order to animate unsteady vector fields smoothly, we place seeds randomly with respect to both time step and position. Then we expand the current time step forward and backward to several time steps. For each time step, we integrate the streamline to create a streamlet corresponding to part of the streamlet object. As we trace we test the canvas to see if a point has been already colored. If the point has been colored the streamlet object is terminated.

According to our experiments, we found if we want to make sure every control point in the streamlet object has not been colored, the failure rate is much higher than dealing with only one time step. The situation becomes worse as the canvas gets full. Usually, it takes a very long time to find a successful streamlet object. We therefore experimented with a more relaxed acceptance criterion, allowing some percentage of the test points to fail. In this case we trace the entire streamlet object and count how many of the test points have been previously colored. A criterion for acceptance or rejection is based on a percentage of the test points that pass the test (acceptance rate). The results showed that a small relaxation in the test criterion greatly improves the success rate, and also increases the density of streamlets in the final rendering.

Once we accept a streamlet object, we add the streamlet to a linked list for later rendering (animation in stage two).

The pseudo code for this algorithm is shown in table 5 and table 6.

Suppose our time varying vector field has N time steps and the size of the vector field is WIDTH by HEIGHT, each 3D streamlet object contains $2*n + 1$ time steps which means that we expand from the selected time step n ($n >= 1$) time steps backward and forward.

```
GenerateSeedPoints(x, y, t)
Randomly select a time step t in the range of 1 to N.
Calculate the beginning (t_begin ) and the ending (t_end) time step for the 3D streamlet object
if (t <= n)
        t_begin = 1
else
        t_begin = t –n

if (t + n > N)
        t_end = N
else
        t_end = t + n
Randomly generate a seed in vector field at position (x, y)

If (advancing seeding position across time steps)
        Advect point (x,y) according to the local vector for each time step in streamlet object
else
        copy position (x, y) as the seeding position for each time step
```

Table 5: Pseudo code for generating seed points for a streamlet object

In the algorithm, we set the value of MAXTRIALS, which means the maximum number of random seeds that are going to be placed. For each 3D streamlet object, valid candidates for control points are recorded and used to compute the success rate. If this rate is larger than the ACCEPTANCERATE, the 3D streamlet object is accepted.

```
WHILE ( number of trials < MAXTRIALS)
{
        GenerateSeedPoints(x, y, t)
        Create a streamlet object which contains t_end – t_start number of streamlets
        number of effective points = 0

        FOR ( start from t_start to t_end )
        {
                Create a streamlet to store positions and vector information
                Convert x_r, y_r to frame buffer coordinate x_r_f, y_r_f based on current time step
                FOR ( seeding forward start from (x_r_f, y_r_f) for streamlet length/2 steps )
                {
                        Add current point to streamlet
                        IF ( glReadPixel(current position) has not been drawn)
                        {
                                number of effective point ++
                        }
                        Move forward based on current vector value and update current point's position
                }

                FOR ( seeding backward start from (x_r_f, y_r_f) for streamlet length/2 steps )
                {
                        Add current point to streamlet
                        IF ( glReadPixel(current position) has not been drawn)
                        {
                                number of effective point ++
                        }
                        Move backward based on current vector value and update current point's position
                }
                Add current streamlet into streamlet array
        }

        IF ( effective point / ((t_end – t_start) * streamlet length) > ACCEPTANCE RATE )
        {
                // MAXFRAMES is interpolation number between time steps
                IF ( start time step >= 1 )
                        Extrapolate random number(less than K) before start time step
                FOR ( start from t_start to t_end )
                        Interpolate MAXFRAMES between two time steps
                IF ( end time step < MAX time step -1 )
                        Extrapolate random number(less than K) after end time step
        }
        Number of trials ++
}
```

Table 6: Pseudo-Code for Generalized Poisson Disk Distribution Algorithm for Unsteady flow Algorithm

3.3.1 Efficiency of Generalized Poisson Disk Distribution Method

Because of the number of tests required for each streamlet object, the overall algorithm can be slow. To speed it up, sub-sampling is used. Canvasses are created only for every Kth time-step and each streamlet, once accepted is interpolated for the intervening K steps, where K >= 1.

The problem with this approach is that streamlets appear and disappear only on the Kth time-step sample points, not at the interpolated time steps, this produces a flickering effect. To alleviate the flicking effect, we extrapolate beyond the first time step and the last time step of the streamlet object by a randomly selected amount in the range of 0 and K. To still further reduce the flickering that occurs when streamlets are born and die, we fade them in an out. This is achieved by varying the transparency of the streamlets.

One of the factors causing poor performance of the algorithm is that testing the frame buffer for a color is a slow operation. It is just as quick to read the entire image back as to test a single point. In order to take advantage of this, many streamlets are seeded and tested in parallel. Each cell was divided into a set of kxk sub cells and a set of pointrs was randomly seeded in each with sufficient spacing that they were unlikely to collide. In addition points were processed at all relevant time steps. This method reduced processing time by more than a factor of 100.

3.4 Results

In this section, we show a selection of results. The basic streamlet shapes adopted in our tests (circle head and arrow head) are selected from the variations we mentioned in section 2.5. In all cases the streamlet's width and length vary proportionally with speed.

We used two datasets to test our algorithm. The first one is an artificially created time varying vector field model since it is easy to control the size and number of time steps. The other one is from the NAM meteorological model obtained from the National Centers for Environmental Prediction. This data has 16 time steps at six hour intervals and a vector field with a 600 * 400 grid.

3.4.1 Artificial Data Model

This dataset was generated with 320 time steps. We subsampled the dataset every 40 time steps for computational efficiency so there are 9 sample time steps in the canvas. The size of the vector field is 300 pixels by 300 pixels. So the size of canvas matrix is 900 pixels by 900 pixels. Each streamlet has 20 control points and the integration step size is proportional to the vector field magnitude with a maximum integration step size of 2 pixels. For each streamlet, W varies with speed (max = 16 pixels), and w varies along the length of the streamlet (1 to W/2). The 3D streamlet object has 5 sample time steps. Because of interpolation and extrapolation, the duration of each streamlet is between 160 time steps and 240 time steps.The animation speed is 50 time steps per second.

Figure 3-5 shows a 3 by 3 canvas matrix used in the first stage of our algorithm. Figure 3-6 shows the actual size of streamlets used for animation. The acceptance rate for a valid 3D streamlet object is 60%, allowing for some overlap.

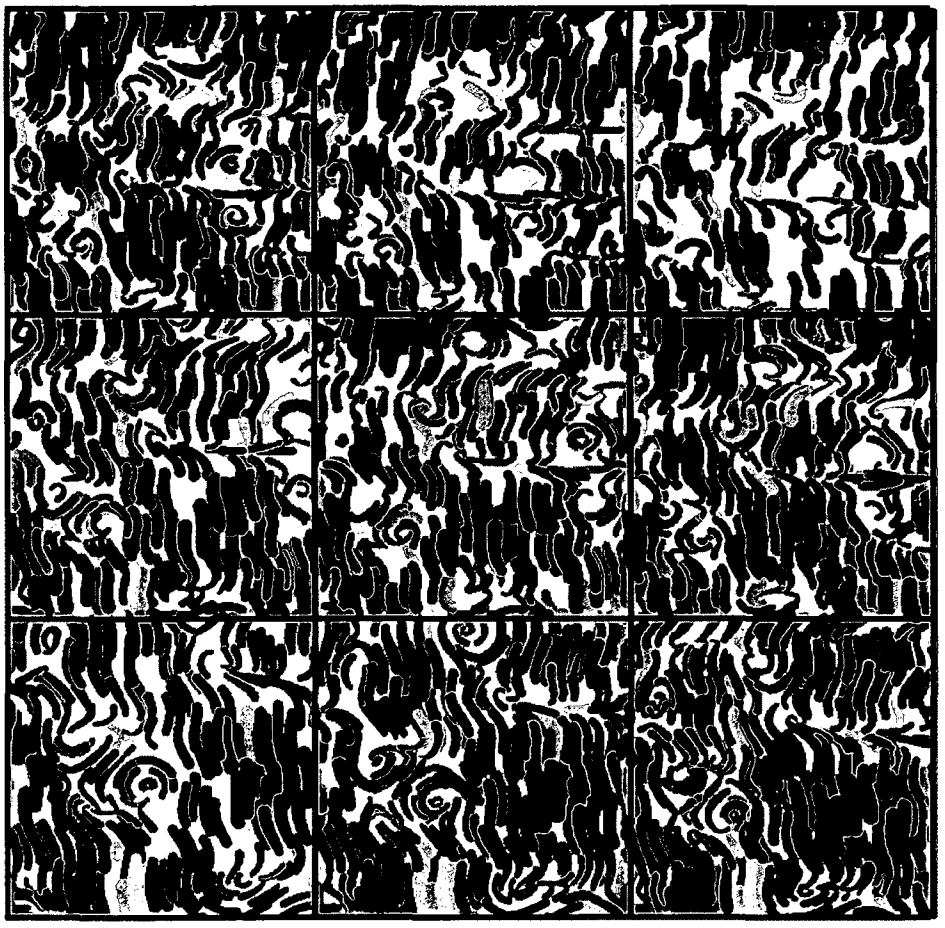


Figure 3-5: 3 by 3 canvas matrix used for image rendering algorithm of unsteady flow (acceptance rate is 60%)

Figure 3-6 shows a subset of the frames from the final rendering. Even though there is some overlap between streamlets, the majority of streamlets are not in contact. We can clearly see the moving path of patterns.

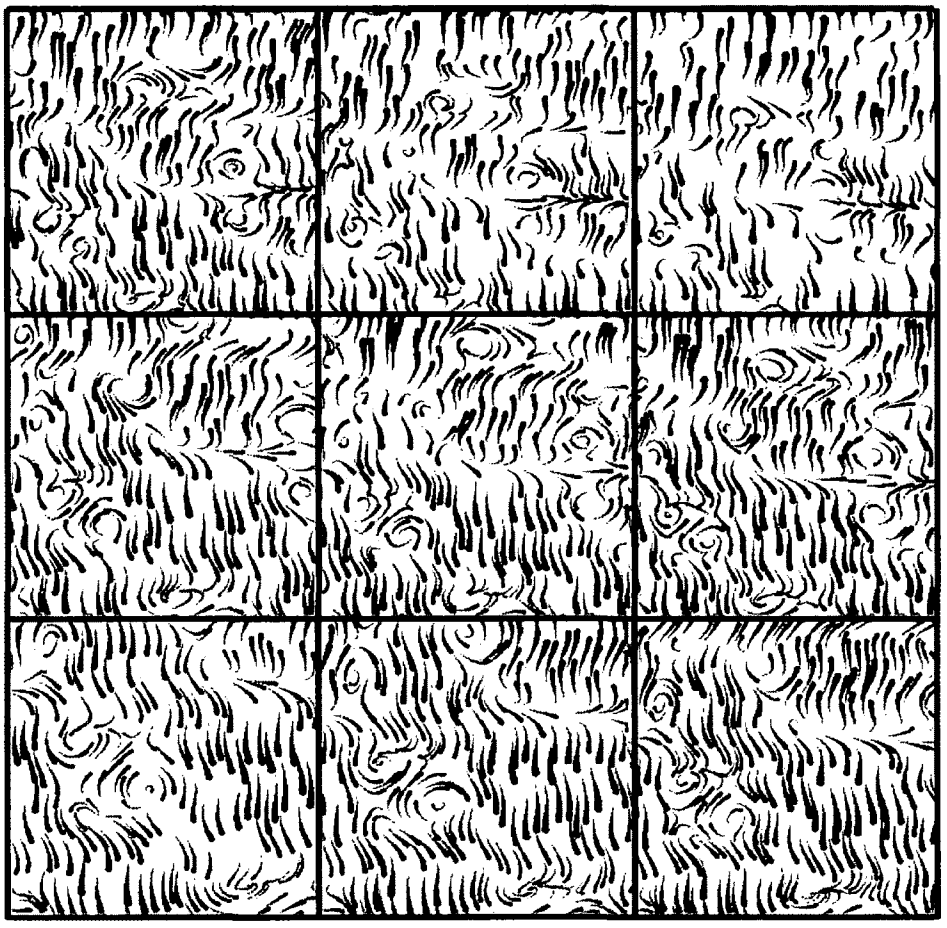


Figure 3-6: Actual size of streamlets used for animation (acceptance rate is 60%)

Figure 3-7 and figure 3-8 show the result of using more restrictive criteria to test a 3D streamlet object than the above examples. The acceptance rate is 90%. The result shows that there is no intersection between streamlets in figure 3-8. Because of the strict criteria, as the space is filled in, most of the tested objects fail resulting in a much lower overall density of streamlets. Comparing the visualization results of those two difference testing criteria, the 60% criteria appears to give a better balance between area coverage and collision rate.

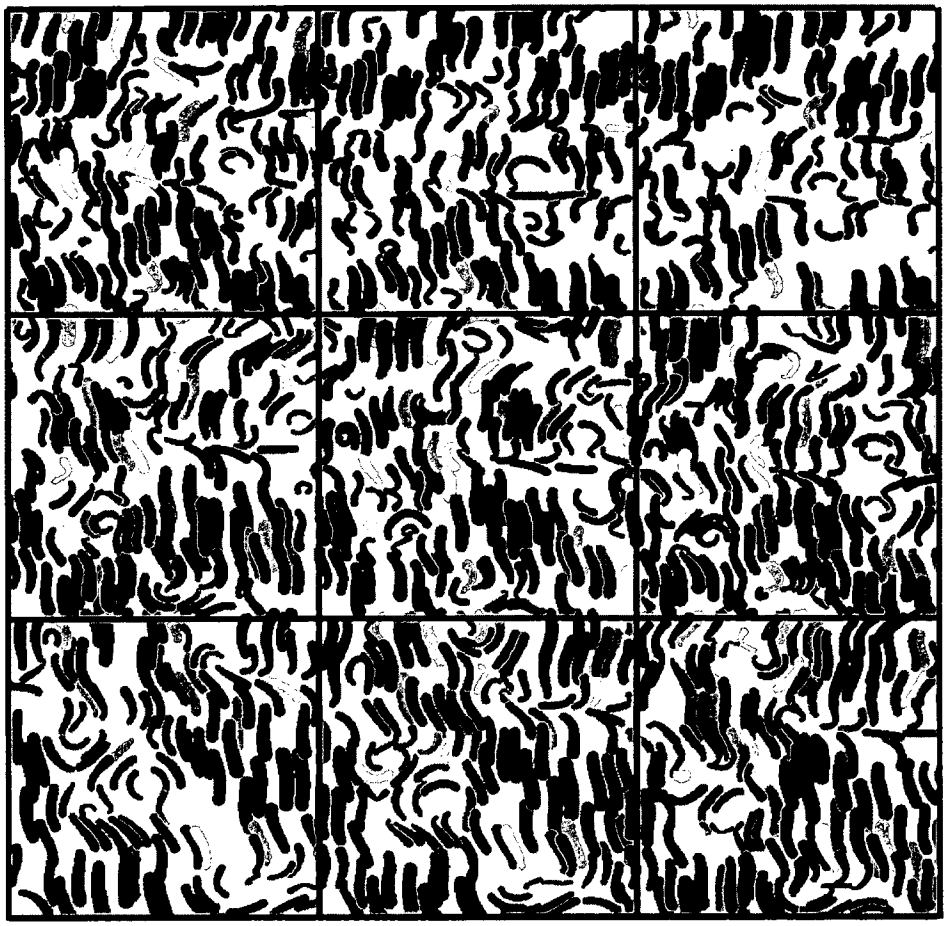


Figure 3-7: 3 by 3 canvas matrix used for image rendering algorithm of unsteady flow (acceptance rate is 90%)

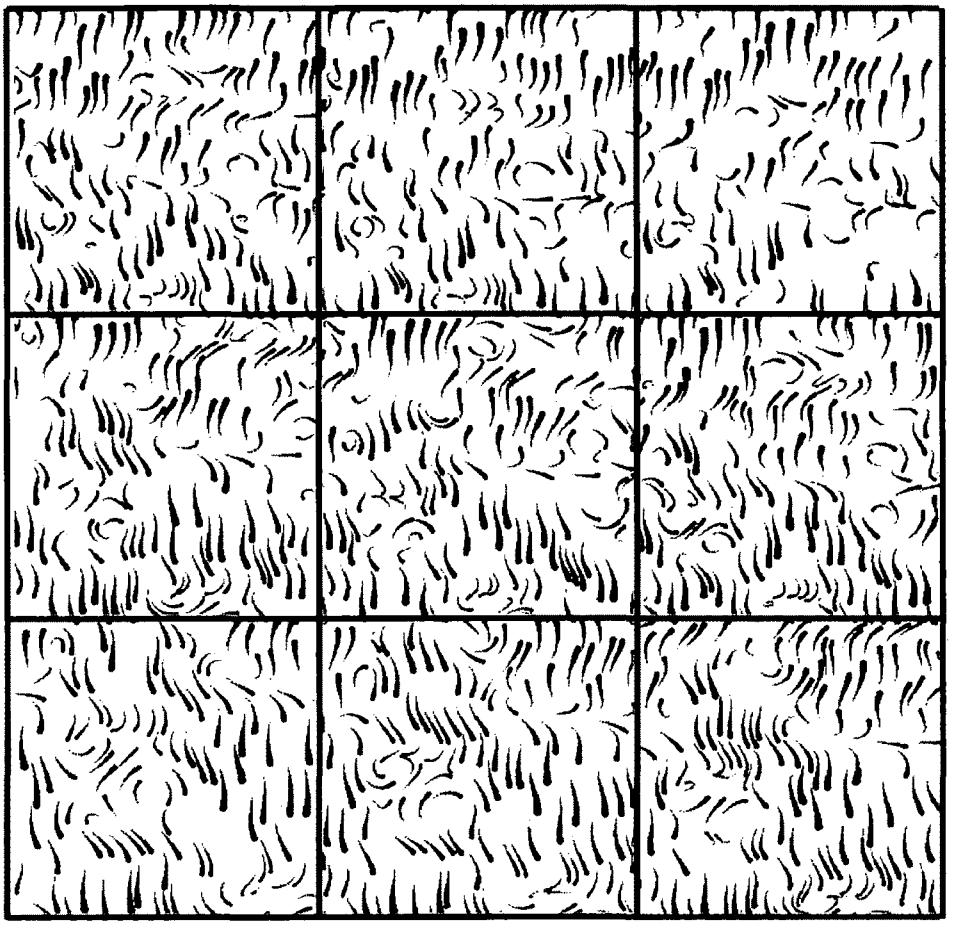


Figure 3-8: Actual size of streamlets used for animation (acceptance rate is 90%)

Figure 3-9 shows a series of snapshots from the interpolated animation for the same example. We can see that consecutive time steps have strong frame to frame coherence. And it is easy to trace the patterns which move smoothly along consecutive time steps.

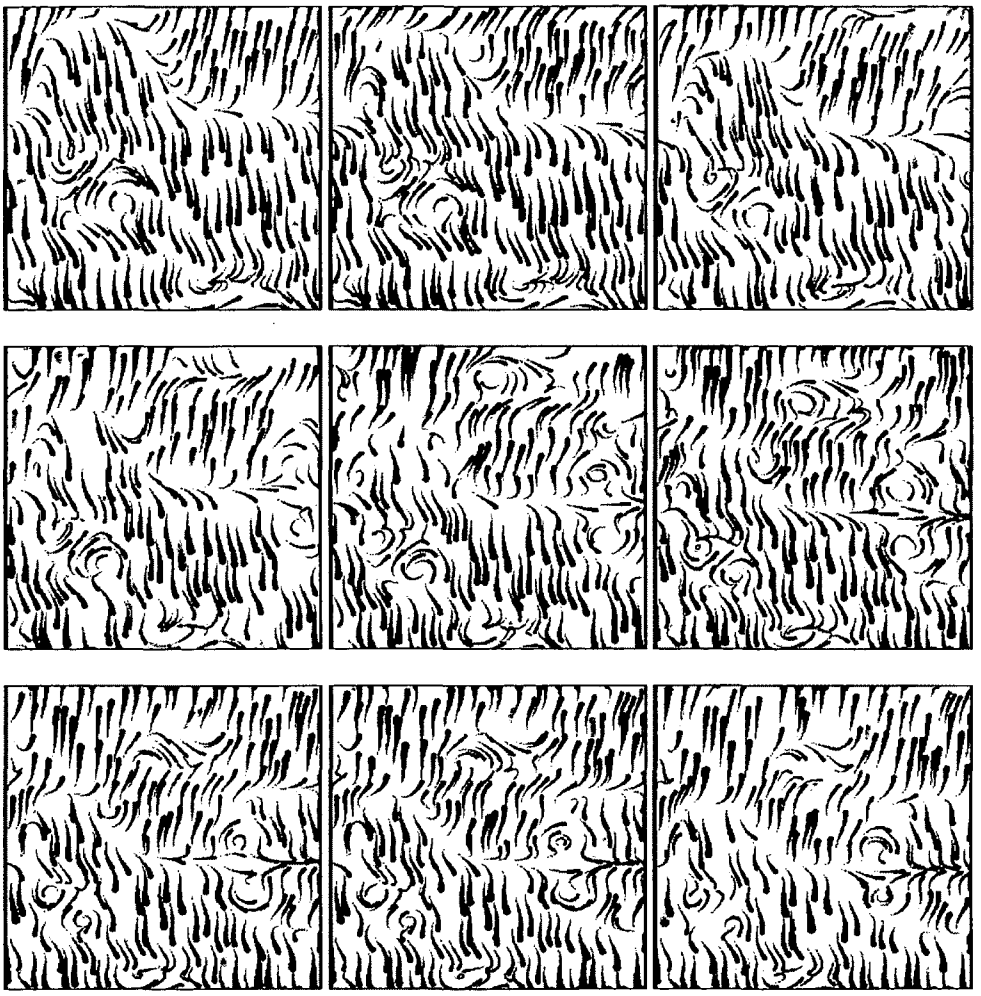


Figure 3-9: Snapshots from unsteady flow animation

Figure 3-10 shows another example. In this case arrow head streamlets are used to rendering a larger unsteady flow (same artificial data model used in above examples). The size of this vector field is 600 pixels by 400 pixels, and the 3D streamlet object acceptance rate is 60%. The rest of the parameters are the same as the above examples. This figure presents more details of the vector field and also shows the evolution of patterns effectively.

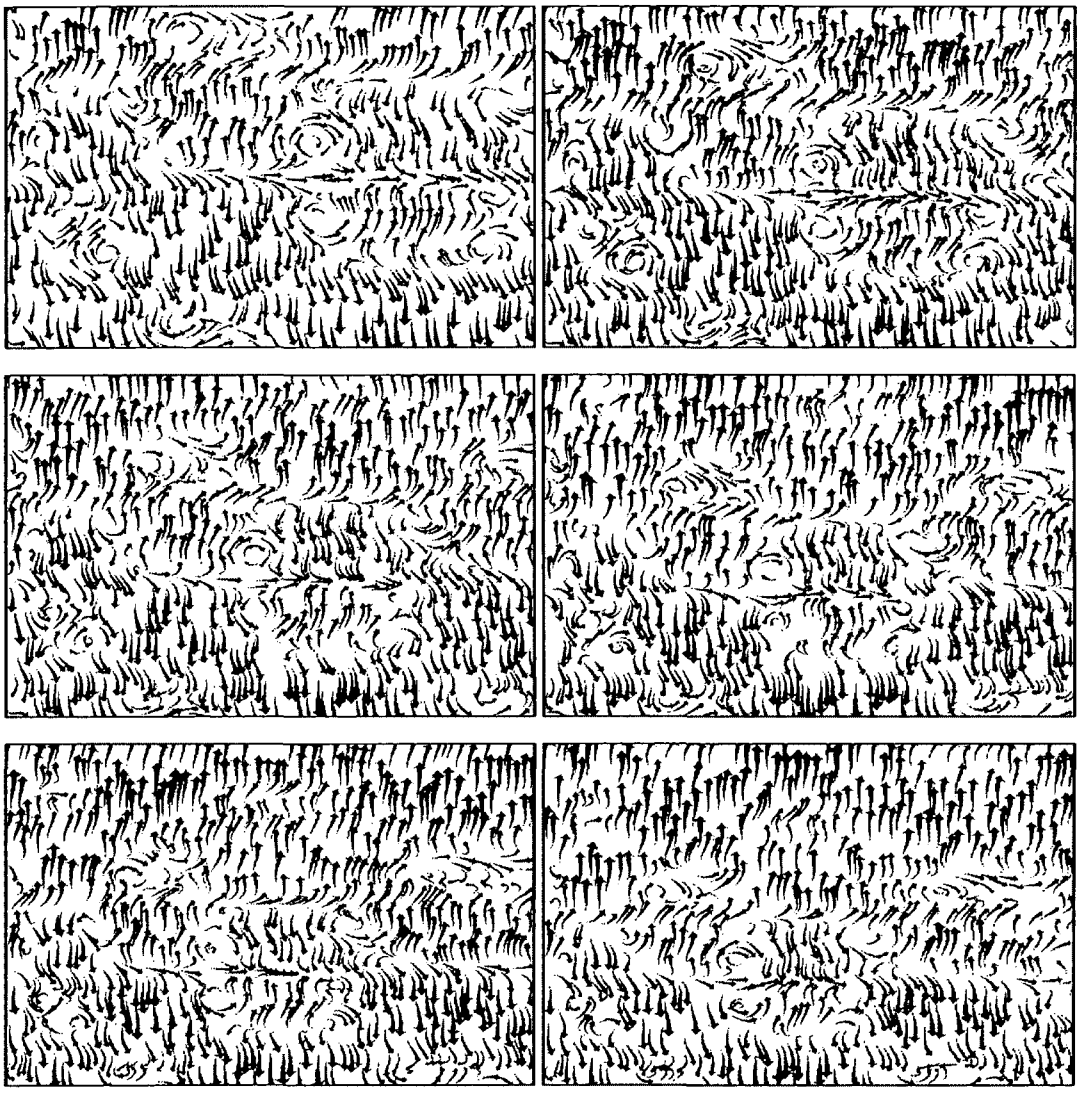


Figure 3-10: Snapshots from unsteady flow animation with arrow head (600 x 400 pixels)

3.4.2 Weather Model Data

The weather model dataset only includes 16 sample time steps, not enough for smooth animation. In the process of generating 3D streamlet objects, we interpolate 40 time steps between successive vector fields if the streamlet object passes the validation test. So we have 640 time steps for animation in total. The size of the vector field is 600

pixels by 400 pixels. Consequently the size of canvas matrix is 2400 pixels by 1600 pixels. Each streamlet has 20 control points and the integration step size is proportional to vector speed (the maximum is 2 pixels). For each streamlet, W varies with speed (max = 16 pixels), and w varies along the length of the streamlet (1 to W/4). The 3D streamlet object has 5 sample time steps. The acceptance rate is 60%. The duration of each streamlet is between 160 time steps and 240 time steps. The animation speed is 50 time steps per second.

Figure 3-11 shows the results using arrow head streamlets to represent vector fields for the 16 sample time steps.

(a) Speed distributions

The speed of the vector field is illustrated by the size of streamlets. From those pictures, we can easily interpret the speed distribution. For instance, in the red rectangle area, the streamlets are bigger than the streamlets in the green circle area. This means the red rectangle area has higher speed than the area identified by the green circle.

(b) Moving pattern (red rectangle)

The area highlighted by the red rectangle shows the winds circulating counter clockwise around a low pressure area. By comparing consecutive time steps, we can tell that the circulation pattern moves in a northerly direction.

(c) At the same position, speed and direction change over time

The areas highlighted by blue rectangles have the same location in all time steps. Comparing these areas we can tell that the speed changes over time (the size of the streamlets keeps changing). This can help us to find a speed changing pattern for a single position.

(d) Direction and orientation

Arrow heads are used to show streamlet direction, and streamlet width increases along the streamlet length as an additional directional cue. For example, it can be seen that wind in the red box area has a very different pattern than wind in the green circle area, where the directions are more randomly distributed.

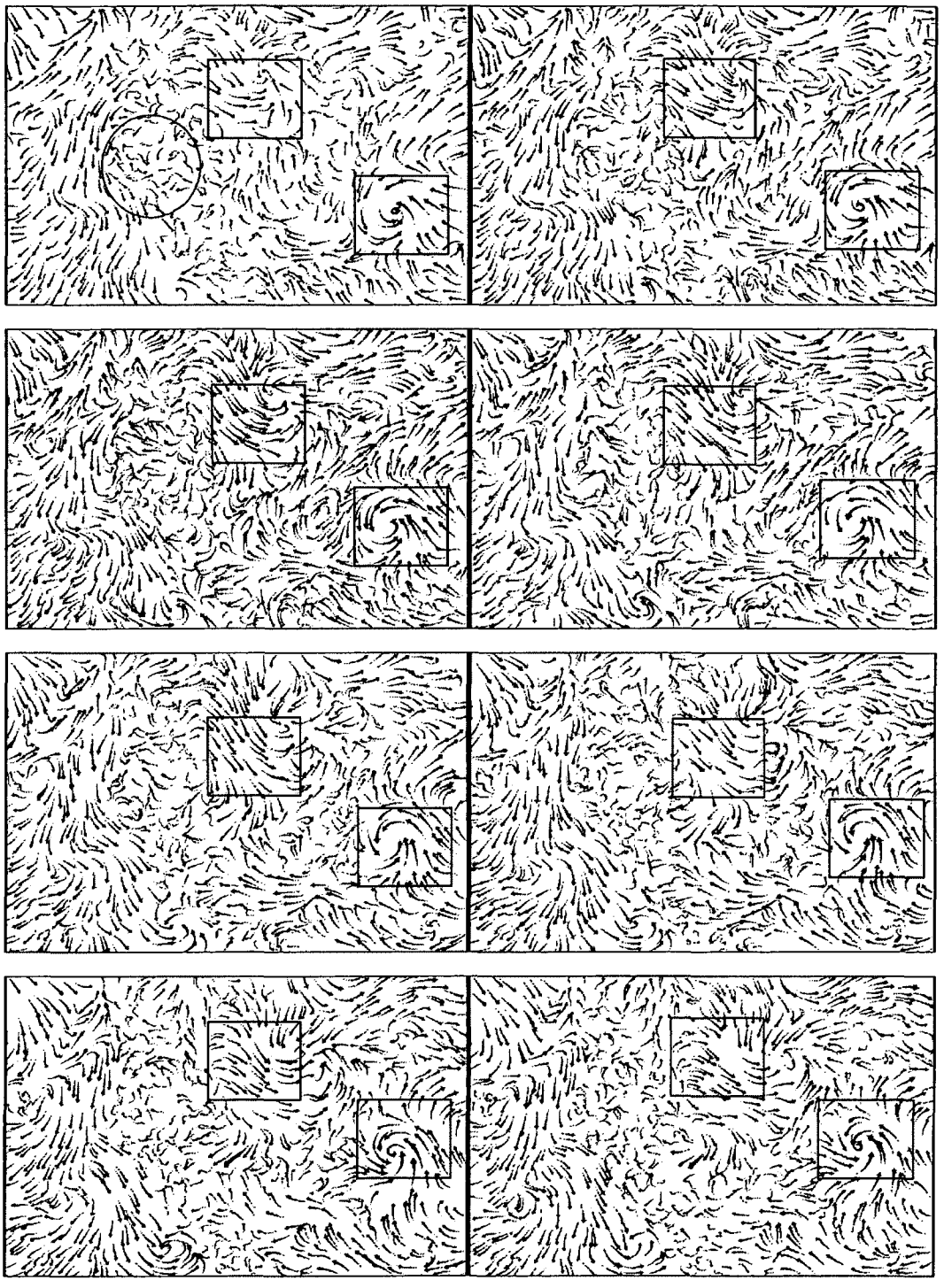


Figure 3-11: 16 time step weather data (600 pixels by 400 pixels)

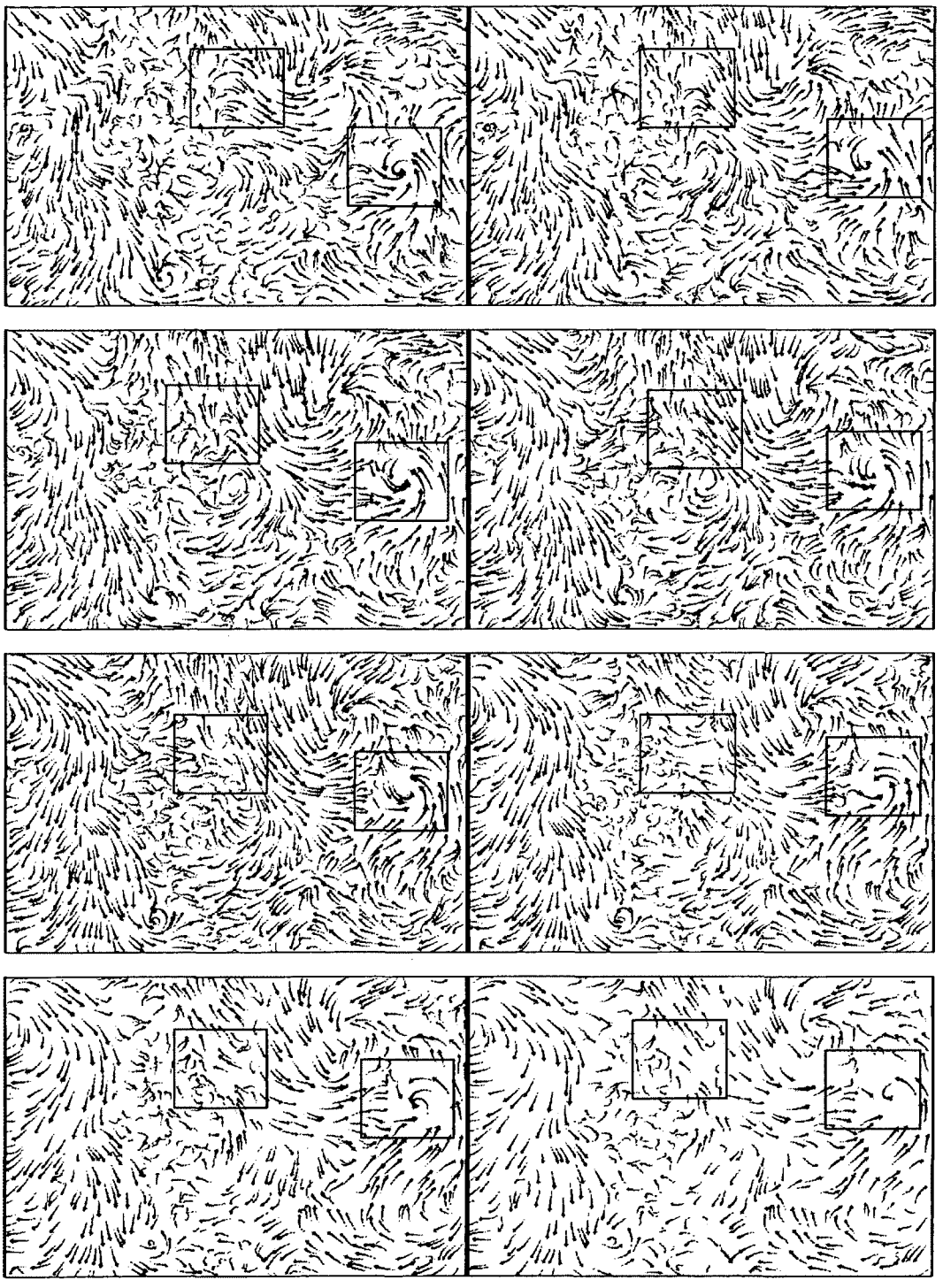


Figure 3-11 (continue): 16 time step weather data (600 pixels by 400 pixels)

CHAPTER 4

CONCLUSIONS AND FUTURE WORK

4.1 Conclusions

The Poisson disk distribution stochastic method was generalized into a technique that can be applied to the problem of flow visualization in order to generate evenly distributed streamlet effects. To increase efficiency over the traditional dart throwing method, an image rendering method was introduced. In this rendering method, every newly generated sample point only has to be compared with one pixel rather than compare with all existing positions, this improves the efficiency dramatically. There are two stages in the basic algorithm. In the first stage circles are drawn sequentially into the frame buffer if the center point has not been previously colored. They are drawn at least two times larger than the intended size. In the second stage the circles are rendered at actual size to generate non-intersecting results. The result shows that the circles are almost evenly distributed in the rendering area without showing any intersections.

Various methods are discussed for generalizing the method to convex shapes other than circles. It is shown that the image rendering algorithm can be used with any convex shapes without intersections as long as the ratio of the size between the drawing shape and the actual shape is larger than the ratio of $(1 + d_{max} / d_{min})$, where d_{max} and d_{min} are the longest and shortest distance from center to edge.

To apply the algorithm to vector field visualizations, the method is expanded to non-convex streamlet shapes suitable for representing steady vector fields. The results show that the method can generate evenly distributed streamlets. By modifying the ratio between a shape's size in two stages, dense or sparse streamlet distribution effects can be achieved. Variations of streamlet shape can be used to get a better representation of flow direction. Those variations include letting the streamlet length and width be proportional to the value of the vector field. Streamlets that taper from head to tail with a circle or arrow head produce very good results in terms of showing direction.

The concept of a 3D streamlet object is introduced for visualizing unsteady flow. The streamlet object correlates consecutive time steps and helps to produce smooth animation results. It is shown that in order to get a balance between intersection and area coverage, the criteria to for acceptance of streamlet objects can be relaxed. Techniques including sub-sampling and parallel seeding strategies improve the speed of the algorithm. Techniques that also improve the appearance of the animation include interpolation, extrapolation, and fading in and fading out of the streamlets.

Both artificially generated data and real weather data were used to test the algorithm. The animation results show that we can clearly observe the moving and changing of flow patterns. The consecutive frames have strong coherence, without showing any flicking or jitters. Other information like speed distribution and flow direction can be easily interpreted from the image.

4.2 Future Work

The method is very flexible. It could be easily extended so that many aspects of a vector field govern the rendering of streamlets. For example, small thin closely spaced streamlets might be used in regions of high vorticity to show detail. Long and sparse streamlets might be used in areas of smooth flow in order to reduce clutter.

The methods described in Chapter 2 for guaranteeing non overlap of symbols often will result in symbols being farther apart than necessary. Future work could investigate the use of different test point schemes to ensure non-overlap of complex non-convex shapes. Figure 4.1 illustrates the concept.

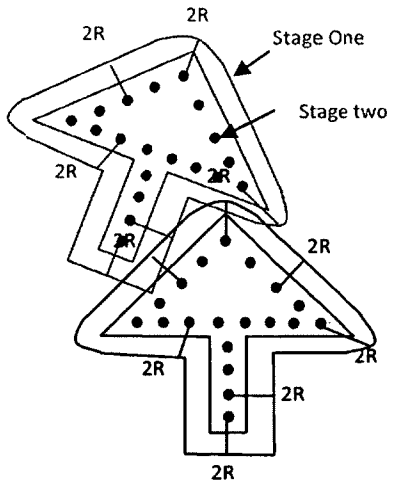


Figure 4-1: The distance from test points to the edge in first stage is two times the maximum distance

More work could also be done on the ideal acceptance rate scheme. It would also be possible to allow streamlets to have greater or lesser duration in order to give more complete coverage of the space time cube.

In addition the results could be evaluated with user studies to determine which schemes are most conducive to the perception of wind patterns. The method also could be evaluated against other techniques for visualizing time varying vector fields, such as animated line integral convolution displays.

For real applications, just showing the flow patterns of vector fields is often not enough. Other information like air pressure usually needs to be shown as well as the wind. Effective methods are needed to show streamlets in combination with other information. These could be implemented and evaluated using methods developed by Mitchell and Ware [1, 23], Laidlaw et al. [7], and others.

BIBLIOGRAPHY

[1] C. Ware. *Toward a perceptual theory of flow visualization.* IEEE Computer Graphics and Applications. 28(2) (2008), pp.6–11.

[2] D. Pineo, and C. Ware. *Neural Modeling of Flow Rendering Effectiveness. ACM Transactions on Applied Perception.* 7(3) (2010).

[3] R. S. Laramee, H. Hauser, H. Doleisch, F. H. Post, B. Vrolijk, D. Weiskopf. *The state of the art in flow visualization: dense and texture-based techniques.* Computer Graphics Forum 23, 2 (2004), 203– 221.

[4] H. Hauser, R. S. Laramee, and H. Doleisch, *State-of-the-art report 2002 in flow visualization.* VRVis Research Center, Vienna, Tech. Rep. TRVRVis-2002-003, 2002.

[5] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. *Over Two Decades of Integration-Based, Geometric Flow Visualization,* Computer Graphics Forum. Volume 29 (2010), number 6, pp. 1807–1829.

[6] Olsen, N. R. B. and Stokseth, S. *Three-dimensional numerical modeling of water flow in a river with large bed roughness,* Journal of Hydraulic Research, Vol. 33, No. 4. (1995), pp. 571-581.

[7] D. H. Laidlaw, M. Kirby, J. S. Davidson, T. Miller, M. DaSilva, W. H. Warren, and M. Tarr. *Quantitative comparative evaluation of 2D vector field visualization methods.* Proceedings IEEE Visualization 2001, pp. 143-150.

[8] R. S. Laramee, H. Hauser, H. Doleisch, F. H. Post, B. Vrolijk, D. Weiskopf. *The state of the art in flow visualization: dense and texture-based techniques.* Computer Graphics Forum 23, 2 (2004), 203– 221.

[9] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, H. Doleisch. *The state of the art in flow visualization: feature extraction and tracking.* Computer Graphics Forum 22, 4 (2003), 775–792.

[10] V. Verma, D. Kao, A. Pang. *A flow-guided streamline seeding strategy.* Proceedings IEEE Visualization 2000 (2000), pp. 163–170.

[11] G. Turk, D. Banks. *Image-guided streamline placement.* ACM SIGGRAPH 96 Conference Proceedings (1996), pp. 453–460.

[12] B. Jobard, W. Lefer. *Creating evenly-spaced streamlines of arbitrary density.* Proceedings of the Eurographics Workshop on Visualization in Scientific Computing '97 (1997), vol. 7, pp. 45–55.

[13] B. Jobard, W. Lefer. *The motion map: efficient computation of steady flow animations.* In Proceedings IEEE Visualization '97 (Oct. 19–24 1997), IEEE Computer Society, pp. 323–328.

[14] B. Jobard, W. Lefer. *Multiresolution flow visualization.* In WSCG 2001 Conference Proceedings Plzen, Czech Republic, February 2001), pp. 33–37.

[15] Z. P. Liu, II R. J. Moorhead. *An advanced evenly spaced streamline placement algorithm.* IEEE Transactions on Visualization and Computer Graphics 12, 5 (2006), 965–972.

[16] D. J. Field, A. Hayes, and R. F. Hess. *Contour integration by the human visual system: Evidence for a local association field,* Vision Research, vol. 33, no. 2, pp. 173 – 193, (1993).

[17] B. Jobard, W. Lefer. *Unsteady flow visualization by animating evenly-spaced streamlines.* Computer Graphics Forum (Eurographics 2000) (2000), vol. 19(3), pp. 21–31.

[18] R. L. Cook. *Stochastic sampling in computer graphics.* Computer Graphics (Proceedings of ACM SIGGRAPH 86) 5, 1 (1986), 51–72.

[19] T. Hachisuka, W. Jarosz, R. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. Jensen. *Multidimensional Adaptive Sampling and Reconstruction for Ray Tracing.* ACM Transactions on Graphics (SIGGRAPH) 27, 3, 33 (2008), 1–33:10.

[20] J. I. Yellott, Jr. *Spectral consequences of photoreceptor sampling in the rhesus retina.* Science 221 (July 22, 1983), 382-385.

[21] G. Kindlmann, C.-F. Westin. *Diffusion tensor visualization with glyph packing.* Proceedings IEEE Visualization 2006 12, 5 (September-October 2006).

[22] L. Feng, I. Hotz, B. Hamann, K. Joy. *Dense Glyph Sampling for Visualization.* Visualization and Processing of Tensor Fields Mathematics and Visualization, 2009, III, 177-193.

[23] Mitchell, P., Ware, C. and Kelley, J. *Designing Flow Visualizations for Oceanography and Meteorology using Interactive Design Space Hill Climbing.* (2009) Proceedings, IEEE SMC.